Optimization Approaches for Mobility and Service Sharing

by

Miao Yu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in the University of Michigan
2020

Doctoral Committee:

Assistant Professor Viswanath Nagarajan, Co-Chair
Associate Professor Siqian Shen, Co-Chair
Professor Marina Epelman
Professor Jon Lee
Professor Yafeng Yin

Miao Yu

miaoyu@umich.edu

ORCID iD: 0000-0002-7625-6315

DEDICATION

This dissertation is dedicated to my wife Kaiwen, for her unending support during my studies, and my dear daughter Euris, for the happiness she brings to my family.

ACKNOWLEDGMENTS

I want to express my special appreciation to my advisor Professor Siqian Shen, who has been a tremendous mentor for me. Thank her for providing insightful supervision on my research and for patiently guiding me to grow as a researcher. I am also very grateful to my co-advisor Professor Viswanath Nagarajan, who guided me into the research world when I first entered graduate school and has always enlightened me with insightful ideas and offered me continuous encouragement. I cannot complete this dissertation without either of them. Besides my advisors, I would like to thank the rest of my dissertation committee: Professor Jon Lee, Professor Marina Epelman, and Professor Yafeng Yin, for their encouragement, insightful comments, and suggestions.

It was also my privilege to learn from professors in the Department of Industrial and Operations Engineering, who built me a solid foundation in Operations Research and enabled me to continue my studies.

I would also like to express my appreciation to Professor Mariel Lavieri, who mentored me when I taught IOE 316 as an instructor, for her continuous supports and suggestions for improving my teaching skills.

I also want to express my thanks to all my colleagues who supported me through my Ph.D. journey: Dr. Yiling Zhang, Hideaki Nakao, Xian Yu, Huiwen Jia, Joy Chang, Qi He, Qi Luo, Dr. Francisco Aldarondo, Zhiyuan Huang, Dr. Yuanyuan Guo, Dr. Xiangkun Shen, Dr. Zheng Zhang, Dr. Yang Zhan, Matthew-Remy Aguirre, Minseok Ryu, Geunyeong Byeon, Haoming Shen, and Zhengtian Xu.

Special thanks to the Michigan Wolverines Football Team for its companionship and inspirations during the past four years.

Last but not least, I am particularly thankful to my family. Words cannot express how grateful I am to my parents for all the sacrifices that they have made for me.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

Mobility and service sharing is undergoing a fast rise in popularity and industrial growth in recent years. For example, in patient-centered medical home care, services are delivered to patients at home, who share a group of medical staff riding together in a vehicle that also carries shared medical devices; companies such as Amazon and Meijer have been investing tremendous human effort and money in grocery delivery to customers who share the use of delivery vehicles and staff. In such mobility and service sharing systems, decision-makers need to make a wide range of system design and operational decisions, including locating service facilities, matching supplies with demand for shared mobility services, dispatching vehicles and staff, and scheduling appointments. The complexity of the linking decisions and constraints, as well as the dimensionality of the problems in the real world, pose challenges in finding optimal strategies efficiently. In this work, we apply techniques from Operations Research to investigate the optimal and practical solution approaches to improve the quality of service, cost-effectiveness, and operational efficiency of mobility and service sharing in a variety of applications. We deploy stochastic programming, integer programming, and approximation algorithms to address the issues in decision-making for seeking fast and reliable solutions of planning and operations problems.

This dissertation contains four main chapters. In Chapter 2, we consider a class of vehicle routing problems (VRPs) where the objective is to minimize the longest route taken by any vehicle as opposed to the total distance of all routes. In such setting, the traditional decomposition approach fails to solve the problem effectively. We investigate the hardness result of the problem and develop an approximation algorithm that achieves the best

approximation ratio. In Chapter 3, we focus on developing an efficient computational algorithm for the elementary shortest path problem with resource constraints, which is solved as the pricing subproblem of the column generation-based approach for many VRP variants. Inspired by the color-coding approach, we develop a randomized algorithm that can be easily implemented in parallel. We also extend the state-of-the-art pulse algorithm for elementary shortest path problem with a new bounding scheme on the load of the route. In Chapter 4, we consider a carsharing fleet location design problem with mixed vehicle types and a restriction on CO_2 emission. We use a minimum-cost flow model on a spatial-temporal network and provide insights on fleet location, car-type design, and their environmental impacts. In Chapter 5, we focus on the design and operations of an integrated car-and-ride sharing system for heterogeneous users/travelers with an application of satisfying transportation needs in underserved communities. The system aims to provide self-sustained community-based shared transportation. We address the uncertain travel and service time in operations via a stochastic integer programming model and propose decomposition algorithms to solve it efficiently.

Overall, our contributions are threefold: (i) providing mathematical models of various complex mobility and service sharing systems, (ii) deriving efficient solution algorithms to solve the proposed models, (iii) evaluating the solution approaches via extensive numerical experiments. The models and solution algorithms that we develop in this work can be used by practitioners to solve a variety of mobility and service sharing problems in different business contexts, and thus can generate significant societal and economic impacts.

CHAPTER 1

Introduction

1.1 Background

Mobility and service sharing is undergoing a fast rise in popularity and industrial growth in recent years (see, e.g., Musich et al., 2015; Shaheen et al., 2015). Such growth is not going to end soon: for the global market of shared mobility, research shows that it is expected to grow from $1.1 billion in 2015 to $6.5 billion by 2024 (Navigant Research, 2017). Via efficient resource pooling, mobility and service sharing has infiltrated people's everyday life and provided practical solutions. For example, in patient-centered medical home care, services are delivered to patients at home, who share medical staff and devices; more and more companies are starting to offer same-day grocery delivery to customers, who share the use of delivery vehicles and staff. In such mobility and service sharing systems, decision-makers need to make a series of decisions, including system design, service planning, and appointment scheduling, which all pose challenges in finding optimal strategies.

One application of mobility and service sharing is a patient-centered medical home. In such a program, routing a fleet of shared vehicles to serve patients requires solving difficult combinatorial problems, while making decisions to achieve a high quality of service needs considerations in optimization under uncertainties. In addition, a large number of different objectives from diverse stakeholders and constraints that ensure valid operations of such systems need to be considered at the same time. Fikar and Hirsch (2017) summarize that

the possible objectives include minimizing traveling cost, operational cost, waiting time, overtime, workload balance, and so on; possible constraints include time windows, skill requirements, working time regulations, breaks, and so on. Up to the present time, a wide variety of optimization models have been developed to address the complicated objectives and constraints when building mobility and service sharing system (see, e.g., Allaoua et al., 2013; Bachouch et al., 2011; Dohn et al., 2009; Fernandez et al., 1974; Lanzarone and Matta, 2014; Zhan et al., 2015).

Carsharing and ride-hailing services are two primary forms in shared-mobility-related business. In these problems, decisions include location design, vehicle routing, demand forecasting, driver-passenger matching, service pricing, and so on (Shaheen et al., 2015). Extensive research work has been conducted to derive optimization models to address aforementioned problems (see, e.g., Boyacı et al., 2015; He et al., 2016; Lu et al., 2018; Nourinejad and Roorda, 2014; Zhang et al., 2018).

In this dissertation, we aim to apply techniques in Operations Research and mathematical optimization to investigate optimal and practical solution approaches for a variety of mobility and service sharing applications, to improve the quality of service, cost-effectiveness, and operational efficiency. In the following, we review some basic models and solution approaches for problems in mobility and service sharing.

1.1.1 Vehicle Routing Problem

A wide range of transportation problems and the related optimization models, especially the ones in mobility and service sharing, are built upon vehicle routing model variants. The vehicle routing problem (VRP) is one of the most classic and well-studied combinatorial optimization problems. Given a set of transportation requests and a fleet of vehicles, VRP aims to find a set of feasible routes (sequence of transportation requests), one for each vehicle, to serve all transportation requests at minimum cost (Toth and Vigo, 2014). Motivated by real-world applications, different variants of VRP have been studied, including Capac-

itated VRP, VRP with time windows, VRP with a heterogeneous fleet, VRP with multiple depots, as well as hybrid versions of these variants, all of which are discussed in detail in Golden et al. (2008) and Toth and Vigo (2014). Each variant introduces a set of constraints that determine the feasibility of routes. For example, in Capacitated VRP, each transportation request is associated with a load, and capacity constraints are enforced so that the total loads over the requests in a route cannot exceed the capacity of the vehicle.

Researchers have extensively studied VRP since it was first proposed by Dantzig and Ramser (1959), who present the first network flow-based mathematical programming formulation and algorithm for capacitated VRP. To optimize VRP exactly, in the early years, the branch-and-cut approach was used to solve the network flow-based model. A class of effective valid inequalities, including capacity, framed capacity, generalized capacity, strengthened comb, multistar, partial multistar, extended hypotour inequalities, and classical Gomory mixed-integer cuts, have been applied to improve the computational efficiency of the branch-and-cut algorithm (Achuthan et al., 2003; Letchford et al., 2002; Lysgaard et al., 2004; Ralphs et al., 2003). Later, Branch-cut-and-price (BCP) became the best-performing exact solution method for VRP after it was first introduced by Fukasawa et al. (2006), in which they combine branch-and-cut with column generation to produce an efficient algorithm for Capacitated VRP instances with up to 135 customers. BCP considers a set partitioning formulation, where decision variables correspond to feasible routes. To avoid explicitly enumerating all feasible routes, it relies on solving pricing problems to find feasible routes that have the potential to be included in the optimal solution. Such pricing problem can be modeled as elementary shortest path problem with resource constraints (ESPPRC), which is often solved using dynamic programming. The BCP approach was later applied to different variants of VRP, and new cutting planes and pricing schemes within BCP were proposed by Baldacci et al. (2010, 2011); Jepsen et al. (2008); Pecin et al. (2017a,b).

1.1.2 Approximation Algorithms

As discussed in Section 1.1, optimization problems in mobility and service sharing involve scheduling, vehicle routing, assignment, and location problems, which have been proven to be NP-hard. Unless $P = NP$, those problems can only be exactly solved for small-sized instances within a reasonable amount of time. An approximation algorithm is an alternative approach, which aims to closely approximate the optimal value of the problem while keeping the runtime polynomial in the problem input size. Williamson and Shmoys (2011) define an α-approximation algorithm as follows.

Definition 1.1. An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.

The approximation guarantee α, also known as the approximation factor or approximation ratio, measures the solution quality of the algorithm. In Chapter 2, we use an approximation algorithm to provide an efficient solution approach to a class of VRPs applicable to the patient-centered medical home problem.

1.1.3 Stochastic Programming

Stochastic programming is one approach to modeling optimization problems whose data incorporated in the objective and constraints is uncertain. The parameters in real-world problems are generally unknown when decisions are made. However, considering the uncertainties is essential in mobility and service sharing systems as decision-makers want to achieve a high quality of service, which is often measured by demand fulfillment rate, customer waiting time, and system overtime. For example, when routing vehicles in a grocery delivery system, the traveling time from one location to another may be affected by weather, traffic, time of the day, and therefore is hard to estimate beforehand; without considering the effect of uncertainties, the routing decision may lead to high overtime for

drivers and long waiting time for customers.

Stochastic programming utilizes the known probability distribution of the data to make an optimal decision (Shapiro and Philpott, 2007). One well-studied stochastic programming model is the two-stage stochastic program, which we apply in Chapter 5 to address the uncertain travel and service time in a routing problem. In the two-stage stochastic programming model, decision-makers make the decision in the first stage before the random events occur in the second stage, in which recourse decisions are made to respond. We optimize the cost of the first-stage decision plus the expected cost of optimal second-stage decisions. An optimal policy to a two-stage stochastic program is a single first-stage decision plus a collection of second-stage decisions that optimally respond to each random outcome.

Let $x \in \mathbb{R}^n$ be the first-stage decision, X be a deterministic feasible region of x, $y \in \mathbb{R}^m$ be the second-stage decision, and $\xi = (q, T, W, h)$ be the random data. Mathematically, a general two-stage stochastic programming problem can be formulated as follows (Birge and Louveaux, 2011):

$$\min_{x \in X} \{g(x) := c^\top x + \mathbb{E}[Q(x, \xi)]\}, \tag{1.1}$$

where $Q(x, \xi)$ is the optimal value of the second-stage problem

$$\min_{y} q^\top y \text{ subject to } Tx + Wy \leq h. \tag{1.2}$$

In the above formulation, ξ is considered as a random vector with a known probability distribution. When ξ has a finite number of realizations, say, $\xi_1, \ldots, \xi_k$, the expectation term in (1.1) can be replaced by $\mathbb{E}[Q(x, \xi)] = \sum_{i=1}^{k} p_i Q(x, \xi_i)$, where p_i is the probability of realization $\xi_i,\ \forall i = 1, \ldots, k$. Then, the two-stage stochastic program (1.1)–(1.2) can be

reformulated as one linear program:

$$\text{minimize: } c^\top x + \sum_{i=1}^{k} p_i q_i^\top y_i \tag{1.3}$$

$$\text{subject to: } x \in X, T_i x + W_i y_i \leq h_i,\ i = 1, \ldots, k. \tag{1.4}$$

Note that the tractability of (1.3) and (1.4) will be affected by the number of realizations of ξ.

To solve (1.1) and (1.2) numerically, we can apply the sample average approximation (SAA) approach (Kleywegt et al., 2002). In SAA, a set of independent and identically distributed samples $\xi_1, \xi_2, \ldots, \xi_k$ is drawn from a given known distribution of ξ, and we can therefore estimate $\mathbb{E}[Q(x, \xi)]$ by $\sum_{i=1}^{k} \frac{1}{k} Q(x, \xi_i)$, where we assume that each sample is realized with equal probability $1/k$. Let $\hat{g}(x) = c^\top x + \sum_{i=1}^{k} \frac{1}{k} Q(x, \xi_i)$ be the approximation of $g(x)$ in Equation (1.1). Kleywegt et al. (2002) show that

$$\min_{x \in X} \hat{g}(x) \to \min_{x \in X} g(x), \text{ with probability 1, as } k \to \infty. \tag{1.5}$$

Furthermore, Kleywegt et al. (2002) show that the probability of SAA approach recovering an optimal solution increases at an exponential rate in the sample size k. In addition, they provide a quantitative bound on the optimality gap, which depends on the sample size k.

The SAA reformulation of the two-stage stochastic programming problem is still computationally expensive due to the size of the problem. Benders decomposition is a decomposition approach that solves large scale linear programs by utilizing their block structures (Benders, 1962). Such an approach can be used to solve stochastic programming problems when the second-stage problem is a linear program (see, e.g., Ahmed, 2013; Birge and Louveaux, 2011; Shapiro et al., 2014). The idea of Benders decomposition is to sequentially construct an approximation of the original problem with a set of cutting planes or cuts. There are two types of cuts: feasibility cuts that determine the feasibility of first-

stage decisions and optimality cuts that approximate the value function of the second-stage recourse problem. Both cutting planes can be generated through solving subproblems. In Chapter 5, we discuss the details of Benders decomposition and apply it as a solution approach to solve a complex stochastic programming problem which arises as a model for VRP with uncertain travel time.

1.2 Dissertation Overview

The remainder of the dissertation contains four main chapters. In Chapter 2, we study a VRP with a minimum makespan objective and compatibility constraints. We provide an approximation algorithm and a nearly-matching hardness of approximation result. We also provide computational results on benchmark instances with diverse sizes showing that the proposed algorithm (i) has a good empirical approximation factor, (ii) runs in a short amount of time, and (iii) produces solutions comparable to the best feasible solutions found by a direct integer program formulation. The work in Chapter 2 has been published in Yu et al. (2017) and Yu et al. (2018).

In Chapter 3, we consider a VRP where each customer has a unit demand representing, e.g., a pick-up or delivery request, and the goal is to minimize the total cost of routing a fleet of capacitated vehicles from depots to serve all customers. We propose two parallel algorithms for the resource-constrained elementary shortest path problem, arising in the pricing problem of the column generation method for solving VRP. The first is an extension of the pulse algorithm by Lozano et al. (2015), for which we derive a new bounding scheme on the load of the route. The second is a randomized algorithm based on the color-coding approach of Alon et al. (1995). The algorithms are general and can be applied to column generation for broader classes of VRPs. We conduct numerical experiments to evaluate the proposed algorithms using modified instances from two different benchmarks, as well as large-scale instances of a patient-centered medical home care delivery problem, based on

census data in Wayne County, Michigan. Our methods are capable of solving the linear programming (LP) relaxation of instances with up to 957 nodes. The numerical results suggest that, in parallel implementations, the random coloring algorithm runs faster than the pulse algorithm when the capacity of the vehicle is small. We also use the generated columns from the LP relaxation to obtain integral solutions. The optimality gap is less than 6% for most of the benchmark instances, and less than 2% for most of the census-data based instances.

Carsharing companies have shown increasing interest in the adoption of fuel-efficient cars to reduce CO_2 emissions and to meet heterogeneous demand. In Chapter 4, we consider location design and relocation problems for sharing a mixed fleet of cars and propose integer linear programs that incorporate both one-way and round-trip demand and operations. To model car movements, we use a minimum-cost flow model on a spatial–temporal network given time-based demand. We maximize the total profit of renting cars minus the cost of relocation and maintenance, subject to limited budget for purchasing cars and given a CO_2 emission limit. In addition, we enforce the first-come, first-served principle to eliminate denied trips. We conduct computational studies based on 2014 Zipcar data in Boston to provide insights for fleet location, car-type designs, and their environmental impacts. Our results show high utilization of cars and low demand losses and denied trips. Although the CO_2 emission limit may lower car-sharing profit, high demand on new energy-efficient cars can compensate the loss and is worth being satisfied. The work in Chapter 4 has been published in Chang et al. (2017).

The fast-growing carsharing and ride-hailing businesses are generating economic benefits and societal impacts in modern society, while both have limitations to satisfy diverse users, e.g., travelers in low-income, underserved communities. In Chapter 5, we study the design and operations of a new car-and-ride sharing system. We consider two types of travelers: Type 1 who rent shared cars and Type 2 who need shared rides. We propose an integrated car-and-ride sharing (CRS) system to enable community-based shared

transportation. To compute solutions, we propose a two-phase approach where in Phase I we determine initial car allocation and Type 1 drivers to accept; in Phase II we solve a stochastic mixed-integer program to match the accepted Type 1 drivers with Type 2 users, and optimize their pick-up routes under a random travel time. The goal is to minimize the total travel cost plus the expected penalty cost of users' waiting and system overtime. We demonstrate the performance of a CRS system in Washtenaw County, Michigan, by testing instances generated based on census data and different demand patterns. We also demonstrate the computational efficacy of our decomposition algorithm benchmarked with the traditional Benders decomposition for solving the stochastic model in Phase II. Our results show high demand fulfillment rates and effective matching and scheduling with low risk of waiting and overtime. The work in Chapter 5 has been published in Yu and Shen (2020).

CHAPTER 2

An Approximation Algorithm for Vehicle Routing with Compatibility Constraints

2.1 Introductory Remarks

VRPs are classic and extensively studied combinatorial optimization problems which aim to find the optimal routing decisions for one or multiple vehicles traveling from the depot(s) to serve demands at various locations. Depending on specific applications, various types of VRPs are formulated and solved by exact or heuristic approaches. We refer the interested readers to Toth and Vigo (2014) and Golden et al. (2008) for comprehensive surveys of models and algorithms for different VRPs, and review the ones that are the most relevant to this chapter below.

A significant amount of VRP literature focuses on single-vehicle VRPs, where only one vehicle is allowed. Without any additional constraints, a basic single-vehicle VRP problem is equivalent to the classic Traveling Salesmen Problem (TSP) (see Applegate et al., 2006). There exist many heuristic approaches, approximations, and exact algorithms for the TSP, and $\frac{3}{2}$ is the best known approximation factor (Christofides, 1976). Some other single-vehicle VRPs are the orienteering problem (Chekuri et al., 2012; Golden et al., 1987), TSP with time windows (Bansal et al., 2004; Barrios and Godier, 2014), Prize-collecting TSP (Balas, 1989), and k-TSP (Garg, 1996).

For multiple-vehicle VRPs, different variants are studied, including multiple TSP (Ma-

lik et al., 2007), capacitated VRP (Chalasani and Motwani, 1999; Charikar et al., 2001), VRP with time windows (Solomon, 1987), and the dial-a-ride problem (Cordeau and Laporte, 2007). The objective in all these studies is to minimize the total traveling cost of all vehicles.

However, with multiple vehicles, another natural objective is to minimize the makespan of the system, i.e., the maximum travel distance among all vehicles. This objective has received relatively less attention (see, e.g., Applegate et al., 2002; Arkin et al., 2006; Even et al., 2004; Gørtz et al., 2016).

In this chapter, we study a multiple-vehicle VRP with a minimum makespan objective where each vehicle can only serve a subset of the locations. Such "compatibility constraints" arise in applications such as medical home care delivery. Here one needs to dispatch shared vehicles to visit patients at their homes. Each patient may require different skill sets from medical teams, who are conveyed by different vehicles, and we also need to balance the workload of different medical staff teams dispatched with the vehicles (Salmond and Ropis, 2005). We note that the work in this chapter has been published in Yu et al. (2017) and Yu et al. (2018).

2.1.1 Problem Definition and Formulation

In this chapter, we study the minimum makespan VRP with compatibility constraints (VRPCC). Given a graph $G = (V, E)$ with node set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i, j) : \ i \in V,\ j \in V\}$, we assume that the depot is located at node 0 and customers are located at the nodes in $V^+ = V \setminus \{0\}$. Each edge $(i, j) \in E$ has a non-negative length c_{ij}, which follows triangle inequality, i.e., $c_{ij} \leq c_{il} + c_{lj}$ for all $i,\ j,\ l \in V$. We assume that the minimum distance is at least 1 and edge lengths are symmetric, i.e., $c_{ij} = c_{ji}$ for all $(i, j) \in E$. A fleet K of vehicles with $K = \{0, 1, \ldots, m-1\}$ is located at the depot, and each can visit a subset of customers in V^+. (Our results also extend easily to the case of multiple depots, but we focus on the single depot case for simplicity.) We assume that

each vehicle $k \in K$ can only visit a subset of nodes $V_k \subset V^+$, based on matches of vehicles and customers' service types. Our goal is to find a routing decision to assign each vehicle a route such that: (a) the nodes visited by vehicle $k \in K$ are in the set V_k; (b) each node must be visited exactly once; and (c) the maximum traveling cost over all vehicles is minimized.

Let $(x_{ij}^k,\ (i,j) \in E)$ be a binary vector of decision variables, such that $x_{ij}^k = 1$ if we assign vehicle $k \in K$ to visit node $j \in V$ right after node $i \in V$, and 0 otherwise. Let $(u_i^k,\ i \in V)$ be the indicator vector parameter, such that $u_i^k = 1$ if $i \in V_k$, and $u_i^k = 0$ otherwise. VRPCC can be formulated as the following integer program.

$$\textbf{MIP: } \underset{\mathbf{x},\,\tau}{\text{minimize}} \ \ \tau \tag{2.1}$$

$$\text{subject to} \sum_{(i,j)\in E} c_{ij} x_{ij}^k \le \tau \quad \forall k \in K \tag{2.2}$$

$$\sum_{(0,j)\in E} x_{0j}^k = 1 \quad \forall k \in K \tag{2.3}$$

$$\sum_{(i,v)\in E} x_{iv}^k - \sum_{(v,j)\in E} x_{vj}^k = 0 \quad \forall v \in V,\ \forall k \in K \tag{2.4}$$

$$\sum_{k\in K} \sum_{(i,j)\in E} x_{ij}^k = 1 \quad \forall j \in V^+ \tag{2.5}$$

$$x_{ij}^k \le u_j^k \quad \forall (i,j) \in E,\ j \in V^+,\ \forall k \in K \tag{2.6}$$

$$\sum_{i,j\in S,\ i\neq j} x_{ij}^k \le |S| - 1 \quad \forall S \subset V^+,\ \forall k \in K \tag{2.7}$$

$$x_{ij}^k \in \{0,1\} \quad \forall (i,j) \in E, \forall k \in K. \tag{2.8}$$

In the above formulation, constraints (2.2) set τ as the maximum traveling cost among all vehicles; constraints (2.3) ensure that each vehicle leaves the depot; constraints (2.4) ensure that the same number of vehicles arrive at and depart from each customer node; constraints (2.5) ensure that each node in V^+ will be visited exactly once; constraints (2.6) ensure that each vehicle can only visit the nodes in its compatibility set; constraints (2.7) are sub-tour elimination constraints avoiding cycles that do not contain the depot; finally, x is a binary decision vector according to constraints (2.8).

Although there are exponentially many constraints in (2.7), this can be addressed by constraint generation as follows. We initially start with no sub-tour elimination constraints in the model. When the solver finds a feasible solution satisfying all the current constraints, we detect cycles that do not pass through the depot. (This can be found very efficiently by a simple graph search.) If we do not find such cycles then the MIP model (2.1)–(2.8) is already solved. Otherwise, we add the corresponding sub-tour elimination constraints from (2.7), and the solver continues solving this new modified model. We iterate this process until a valid solution is found.

Despite the successful usage of column generation-based algorithms in VRP, our preliminary study shows that solving MIP model by commercial solver outperforms the column generation-based algorithm for VRPCC problem (Yu et al., 2017).

In our computations, we solve this MIP model (2.1)–(2.8) using a state-of-the-art solver (Gurobi), and compare its performance to our approximation algorithm. We also use MIP to compute lower bounds for VRPCC.

Recall that an α-approximation algorithm for a minimization (resp., maximization) problem always produces a solution of objective value at most (resp., at least) α times the optimal.

2.1.2 Main Results

In this chapter, we focus on the approximability of the VRPCC.

Theorem 2.1. VRPCC cannot be approximated to within a factor of $(1-o(1))\cdot\ln n$, unless $\mathcal{NP}=\mathcal{P}$.

Theorem 2.2. There is a $2\lceil\ln n\rceil+1$-approximation algorithm for VRPCC.

Both the hardness result and the algorithm are based on relations to the set cover problem, which is known to have a tight approximability threshold of $\ln n$ (see Chvatal, 1979; Dinur and Steurer, 2014). Recall that the set cover problem involves selecting the smallest

number of sets from a given collection so as to cover all elements of a ground set. The main idea for Theorem 2.1 is to reduce the min-sum objective in set cover to the min-max objective in VRPCC. This is done by making several copies of the set cover instance and "rotating" the set-element relations in each of these instances so as to balance the load across all sets used in an optimal solution. See Section 2.2 for details.

Theorem 2.2 is based on applying the set cover greedy algorithm on an implicit set system; such an approach has been used in a number of approximation algorithms, e.g., Klein and Ravi (1995); Svitkina and Tardos (2004). However, the "max coverage" subproblem that we need to solve is different in order to handle the min-max objective. This corresponds to the maximum coverage problem with group budgets (Chekuri and Kumar, 2004; Martin and Salavatipour, 2017) (we will define it formally later), and we can apply their approach using an approximation algorithm for the orienteering problem (Chekuri et al., 2012). We observe that using an approximation algorithm for the related k-TSP problem (Garg, 2005) leads to a slightly better constant (2 instead of $2 + \epsilon$) in the approximation ratio, but more importantly this approach is far easier to implement. See Section 2.3 for details.

Our computational results show that the empirical approximation factor (solution value from the approximation algorithm divided by the best lower bound found) is much smaller than the theoretical one. Moreover, the running time of the approximation algorithm is much smaller than MIP: the time taken is less than 2 minutes on almost all instances. Also, the solution found by our approximation algorithm is very close (or better) than the best solution found by MIP even after 2 hours. See Section 2.4 for details.

2.1.3 Related Work

For VRPs with a min-max objective, most current literature focuses on heuristic methods (see, e.g., Carlsson et al., 2009; Golden et al., 1997) and approximation algorithms (see, e.g., Arkin et al., 2006; Even et al., 2004; Gørtz et al., 2016). In particular, constant-factor approximation algorithms are known for min-max (unrooted) path/tour cover (Arkin

et al., 2006), min-max (rooted) path/tour cover (Even et al., 2004) and min-max TSP with non-uniform speeds (Gørtz et al., 2016). To the best of our knowledge, approximation algorithms for min-max VRPs with compatibility constraints have not been studied, and this chapter aims to fill this gap. Compared to these previous results, we show that the problem with compatibility constraints does not admit any constant-factor approximation.

In this chapter, we use a set cover-based technique to derive our algorithm. Set cover and maximum coverage problems are classic problems in combinatorial optimization with wide applications in various settings. A greedy algorithm that repeatedly picks the set covering the maximum number of uncovered elements yields a $(\ln n)$-approximation for the set cover problem and an $\frac{e}{e-1}$-approximation for the maximum coverage problem (see Chvatal, 1979). This is also known to be best-possible unless $\mathcal{NP} = \mathcal{P}$ (Dinur and Steurer, 2014). Our algorithm relies on the framework of maximum coverage with group budgets, introduced by Chekuri and Kumar (2004); the approximation ratio was recently improved in Martin and Salavatipour (2017). A crucial subroutine in implementing this framework for VRPCC is the k-TSP problem, which finds a rooted tour with minimum total cost covering k nodes in a given graph. A constant-factor approximation algorithm for k-TSP was given by Blum et al. (1996) and later improved by Garg (1996), Arora and Karakostas (2006), and Garg (2005) to 3, $2 + \epsilon$, and 2, respectively.

2.2 Hardness of Approximation for VRPCC

We prove Theorem 2.1 by showing the set cover problem is polynomial-time reducible to VRPCC, and therefore VRPCC is at least as hard as the set cover problem.

Proof of Theorem 2.1. In a set cover instance, we are given a ground set $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$. A cover is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets whose union is $\mathcal{U}$. The objective is to find a cover that uses the fewest sets in $\mathcal{S}$. Let $[t]$ denote the integer set $\{0, 1, \ldots, t-1\}$ for any $t \geq 1$.

Given an input $(\mathcal{U}, \mathcal{S})$ of set cover, we index elements in $\mathcal{U}$ as $0, 1, \ldots, n-1$ and let the collection of subsets be $\mathcal{S} = \{S_i : i \in [m]\}$. The reduction constructs a graph with $mn + 1$ nodes that are partitioned into a depot node r and m disjoint groups $W_i = \{u_{ij}, j \in [n]\}$ for each $i \in [m]$. There are edges of cost 0 between each pair of nodes in the same group $\{W_i\}_{i \in [m]}$ and an edge of cost 0.5 between r and all other nodes. There are no edges between nodes of different groups. Let T denote this edge-weighted graph and $c_{a,b}$ the shortest path distance between nodes a and b in T. Figure 2.1 shows an example of graph T with $n = 5$ and $m = 4$.

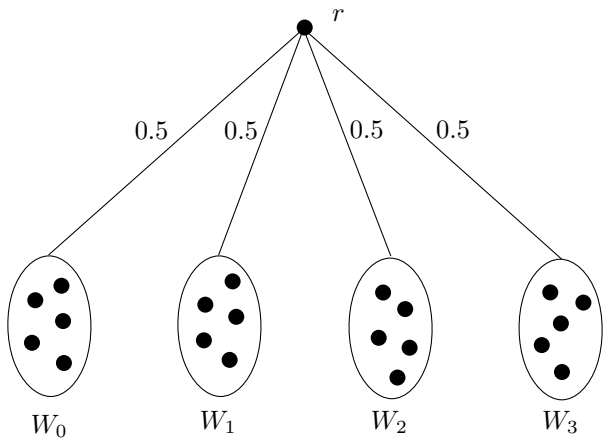
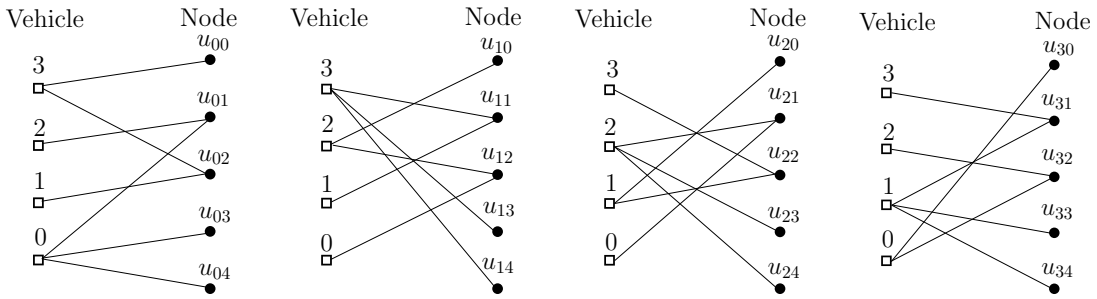


Figure 2.1: An example of T with $n = 5$, $m = 4$. Each node in the graph is connected to r with edge cost 0.5 and in each group $\{W_i : i \in [4]\}$ the nodes comprise a complete graph with edge costs 0.

The VRPCC instance has m vehicles, so $K = [m]$. The compatibility constraints $\{V_k : k \in [m]\}$ are based on "rotating" the sets in $\mathcal{S}$ and are defined as follows. For any vehicle $k \in [m]$, group $i \in [m]$ and $j \in [n]$, node $u_{ij} \in V_k$ if and only if $j \in S_{k'}$, where $k' = (k + i) \mod m$. Figure 2.2 shows an example for an instance with $m = 4$ and $n = 5$ where the relationship in W_0 represents the original collection $\mathcal{S}$. This reduction is clearly polynomial time in m and n.

We argue that solving the set cover instance $(\mathcal{U}, \mathcal{S})$ is equivalent to solving the above VRPCC instance. Let SC^* and CC^* denote corresponding optimal solutions to set cover and VRPCC, respectively. Let $c(SC^*)$ and $c(CC^*)$ denote corresponding optimal objec-



(a) Group W_0 (b) Group W_1 (c) Group W_2 (d) Group W_3

Figure 2.2: An illustration of compatibility constraints on different groups.

tives.

We first show that $c(CC^*) \leq c(SC^*)$. Let C contain indices of all sets in SC^*. Then, we construct the following solution to VRPCC. We define $C_i := \{(c-i) \mod m \mid c \in C\}$ for each group $i \in [m]$. Note that by definition of the compatibility constraints in group i, the vehicles in C_i can cover all nodes in W_i, and so the VRPCC solution involves routing each vehicle $k \in [m]$ to the groups $\{i \in [m] : k \in C_i\}$. The total cost for any vehicle $k \in [m]$ is then $|\{i \in [m] : k \in C_i\}| = |C|$, so the VRPCC objective is also $|C| = c(SC^*)$. This implies $c(CC^*) \leq c(SC^*)$.

Conversely, we show that $c(SC^*) \leq c(CC^*)$. Consider the optimal VRPCC solution CC^*. As $c(CC^*)$ is the maximum cost over all m vehicles, the total cost to cover all nodes is at most $m \times c(CC^*)$. This implies that there exists a group, say, W_i, which is visited by *at most* $c(CC^*)$ vehicles. Let $D_i \subseteq [m]$ denote the set of vehicles that visit group W_i; note that $|D_i| \leq c(CC^*)$. Then it follows (due to the compatibility constraints for W_i) that $D = \{(d+i) \mod m \mid d \in D_i\}$ forms a valid set cover, i.e. $\cup_{\ell \in D} S_\ell = \mathcal{U}$. So, the optimal set cover value $c(SC^*) \leq |D| = |D_i| \leq c(CC^*)$.

Combining both, we conclude that $c(CC^*) = c(SC^*)$. The result in Theorem 2.1 now follows from the hardness for set cover that there is no $(1 - o(1)) \cdot \ln n$ approximation algorithm unless $\mathcal{P} = \mathcal{NP}$ (Dinur and Steurer, 2014). This completes the proof. □

2.3 An $O(\log n)$-Approximation Algorithm

In this section, we propose an approximation algorithm for VRPCC. Our algorithm starts with a "guess" B on the optimal value, which can be verified within a binary search scheme (see Algorithm 2.2). Then it iteratively picks a route for each vehicle that covers (approximately) the maximum number of new nodes. We iterate this process until all nodes are covered.

We start by introducing the maximum coverage problem with group budgets (MCG). Here, we are given a ground set for MCG, X, and a collection of subsets of X, $\mathcal{C} = \{S_1, S_2, \ldots, S_m\}$. We are also given a partition of $\mathcal{C}$ into k groups $G_1, G_2, \ldots, G_k$. A solution to MCG is a subset $H \subset \mathcal{C}$ such that $|G_i \cap H| \leq 1$ for all $i = 1, \ldots, k$, i.e., at most one set can be chosen from each group. The objective is to maximize the number of elements in X covered by H.

Example: Consider $X = \{1, 2, 3, 4, 5\}$, $S_1 = \{1, 2, 3\}$, $S_2 = \{4, 5\}$, $S_3 = \{1, 2, 5\}$, $S_4 = \{1, 5\}$ and $k = 2$. Suppose that $G_1 = \{S_1, S_3\}$ and $G_2 = \{S_2, S_4\}$. Then the optimal solution is $H = \{S_1, S_2\}$ which covers all 5 elements.

Chekuri and Kumar (2004) gave a greedy algorithm to solve MCG by iteratively picking one set from each group that covers the maximum number of uncovered elements. Crucially, the algorithm works with an oracle model $\mathcal{O}$ that takes as input a ground set X' and an index i and outputs a set $S_j \in G_i$ such that $|S_j \cap X'|$ is maximized. They showed that this greedy algorithm is a $\frac{1}{1+\rho}$-approximation algorithm for MCG given a $\frac{1}{\rho}$-approximate oracle. Later, Martin and Salavatipour (2017) proposed a better $(1 - e^{-1/\rho})$-approximation algorithm, given a $\frac{1}{\rho}$-approximate oracle, based on a linear program rounding algorithm. However, this algorithm relies on using the ellipsoid method to solve LPs, which is not practical for our purpose.

Specializing MCG to the VRPCC setting, we obtain the following.

Problem 2.1. MCG-VRP

Input: a node subset $X \subset V$, a fleet K of vehicles and a budget $B \geq 0$.

Output: routes $\{A_i \subseteq V_i : i \in K\}$ for each vehicle with each route of cost at most B.

Objective: maximize $|\cup_{i \in K} A_i \cap X|$, the number of nodes covered.

In this case, the oracle $\mathcal{O}$ corresponds to the orienteering problem. Formally, $\mathcal{O}(Y, B, i)$ involves computing a route for vehicle $i \in K$ (originating from r) with cost at most B that covers the maximum number of nodes in Y. The compatibility constraints are enforced in the definition of this oracle instance. The complete algorithm is described in Algorithm 2.1.

Algorithm 2.1: Greedy Algorithm for MCG-VRP

input : A fleet K of vehicles, a subset $X \subset V$ and a budget B
output: A set H of routes with cost at most B, one route for each vehicle

```
X' ← X
for i ∈ K do
    A_i = O(X' ∩ V_i, B, i)
    X' ← X'\A_i
end
return H = {A_i : i ∈ K}
```

Theorem 2.3 (Corollary 1 in (Chekuri and Kumar, 2004))**.** *Algorithm 2.1 is a $\frac{1}{1+\rho}$-approximation algorithm for the MCG-VRP, assuming a $\frac{1}{\rho}$-approximate oracle $\mathcal{O}$.*

The current best approximation ratio for the orienteering problem is $(2 + \epsilon)$ with running time $n^{O(1/\epsilon^2)}$ (Chekuri et al., 2012); so this is polynomial time only for constant $\epsilon > 0$. However, this algorithm as well other constant-factor approximation algorithms for orienteering are rather complex to implement. To simplify the implementation, we instead use a $(1, \beta)$-*bicriteria* approximation algorithm for orienteering, which violates the cost by a factor $\beta \geq 1$ but covers the optimal number of nodes (for a cost B route). This corresponds to the k-TSP problem: given a graph with root r and target k, find a min-cost route (originating from r) that covers at least k nodes. Given a β-approximation algorithm for k-TSP, we can obtain a $(1, \beta)$-*bicriteria* approximation for orienteering by just running a binary search on k to output the route having highest k and cost at most βB. The best

approximation ratio for k-TSP is $\beta = 2$ from Garg (2005). This algorithm is also much more efficient than those for orienteering; in fact, it is used as a subroutine in all algorithms for orienteering. Using Theorem 2.3, we obtain:

Corollary 1. If we use a $(1, \beta)$-*bicriteria* approximation algorithm for oracle $\mathcal{O}$, then Algorithm 2.1 is a $(\frac{1}{2}, \beta)$-*bicriteria* approximation algorithm for MCG-VRP.

Finally, we use Algorithm 2.1 iteratively until all nodes are covered, and perform a binary search on the "guess" B. The details are displayed in Algorithm 2.2.

Algorithm 2.2: Approximation Algorithm for VRPCC

input : A network $G = (V, E)$, a fleet K of vehicles
output: Routing assignment for each vehicle in $i \in K$

1 Initialize routes $\tau_i = \emptyset$ for all $i \in K$, $X \leftarrow V^+$.
2 Initialize upper bound $u = 2\sum_{(i,j)\in E} c_{ij}$, lower bound $l = 0$, and a tolerance threshold ϵ for the binary search.
3 **while** $u - l \geq \epsilon$ **do**
4 $\quad B \leftarrow \frac{u+l}{2}$, $Solve \leftarrow$ **true**
5 $\quad$ **while** $X \neq \emptyset$ **do**
6 $\quad\quad \{A_i : i \in K\}$ = solution from Algorithm 2.1 for MCG-VRP with input K, X, B
7 $\quad\quad$ **if** $|\cup_{i\in K} A_i \cap X| < |X|/2$ **then** $Solve \leftarrow$ **false**, **break**
8 $\quad\quad$ Update $X \leftarrow X \setminus (\cup_{i\in K} A_i)$ and $\tau_k \leftarrow \tau_k \circ A_k$ for all $k \in K$
9 $\quad$ **end**
10 $\quad$ **if** $Solve$ **then** $u \leftarrow B$
11 $\quad$ **else** $l \leftarrow B$
12 **end**
13 **return** routes in τ_i for $i \in K$

The following lemma shows VRPCC can be solved by iterating Algorithm 2.1 at most $\lceil \log_2 n \rceil$ times for the correct guess of B.

Lemma 2.1. If we use a $(1, \beta)$-bicriteria approximation algorithm for oracle $\mathcal{O}$ then Algorithm 2.2 achieves a $(1 + \epsilon)\beta\lceil \log_2 n \rceil$ approximation ratio for VRPCC.

Proof. By the definition of upper/lower bounds (u and l) and the binary search on B, it is clear that the parameter $Solve$ is true (resp., false) at the end of the inner while-loop (line 9)

when B is set to u (resp., l). Note also that for any B where $Solve$ is true at the end of the inner while-loop, Algorithm 2.1 must have produced (in each iteration of the inner while-loop) a solution that covers at least half of the current set X: as each iteration increases makespan by at most βB (by Corollary 1), the resulting VRPCC solution has makespan at most $(\beta\lceil\log_2 n\rceil)\cdot B$. In particular, for the choice $B = u$ at the end of the algorithm, we obtain that the makespan of Algorithm 2.2 is $ALG \leq \beta\lceil\log_2 n\rceil \cdot u$.

Suppose the optimal value for VRPCC is B^* and consider any $B \geq B^*$. Note that the optimal value of the MCG-VRP instance (X, B, K) is $|X|$ for any $X \subseteq V^+$; hence, using Corollary 1, Algorithm 2.1 covers at least half the nodes in X. So, $Solve$ is true at the end of the inner while-loop for any $B \geq B^*$. As $Solve$ is false at the end when $B = l$, we obtain $B^* \geq l$. So we have:

$$ALG \leq \beta\lceil\log_2 n\rceil \cdot u \leq \beta\lceil\log_2 n\rceil(l+\epsilon) \leq \beta\lceil\log_2 n\rceil(B^*+\epsilon) \leq (\beta\lceil\log_2 n\rceil)(1+\epsilon)B^*.$$

The last inequality uses the fact that $B^* \geq 1$ as all distances are assumed to be at least one. Therefore, Algorithm 2.2 is a $(1+\epsilon)\beta\lceil\log_2 n\rceil$-approximation algorithm. □

Using $\beta = 2$ and $\epsilon \leq \frac{1}{n}$, we obtain a $2\lceil\log_2 n\rceil+1$-approximation algorithm for VRPCC. If, instead of the greedy approach from Chekuri and Kumar (2004), we use the LP-based approach in Martin and Salavatipour (2017) to solve MCG-VRP, we will obtain a $(2\ln n+1)$-approximation algorithm. This completes the proof of Theorem 2.2.

For the computational results, we only tested the greedy approach which has a slightly worse approximation ratio, but is a lot simpler to implement.

The time complexity of Algorithm 2.2 depends on the complexity of the oracle $\mathcal{O}$. Suppose the complexity of $\mathcal{O}$ is $T(n)$; then the complexity of Algorithm 2.1 is $O(mT(n))$. For Algorithm 2.2, the inner while-loop executes at most $\lceil\log n\rceil$ as we halve the size of X in each loop. The outer loop is used to conduct a binary search for the optimal budget, which takes $\log_2 \frac{C}{\epsilon}$ steps, where $C = 2\sum_{(i,j)\in E} c_{ij}$ is a trivial upper bound. Therefore, the

overall complexity of Algorithm 2.2 is $O\left(\log_2(\frac{C}{\epsilon}) \cdot \log_2(n) \cdot m \cdot T(n)\right)$.

2.4 Computational Results

To evaluate the performance of the proposed approximation algorithm, we conduct numerical studies on tailored instances generated from Solomon's benchmark for VRPs (Solomon, 1987). First, we compare the proposed approximation algorithm against the MIP on small instances with up to 25 customers. Then, we extend our experiments to instances with up to 100 customers.

There are three categories of the Solomon's instances based on the nodes distribution: the R type where nodes are uniformly randomly generated on the grid, the C type where nodes are distributed in clusters, and the RC type that is a mixed of R type and C type. We only use the node location from Solomon's benchmark and customize our instances as follows. We sample the desired number of nodes from the benchmark instances and compute the distance matrix. We consider two types of VRPCC instances: one with tight compatibility constraints where each node can be visited by a small number of vehicles, and the other with relaxed compatibility constraints where each node can be visited by a larger number of vehicles. In our tailored instances, we randomly generate compatibility constraints such that we allow one vehicle to visit a node with 30% probability for tight instances and 70% probability for relaxed instances. Each instance is labeled by its type R/C/RC, number n of nodes, and number k of vehicles.

To implement the approximation algorithm, we use the 5-approximation algorithm for k-TSP problem from Garg (1996) to serve as our oracle $\mathcal{O}$. Although the best known result is a 2-approximation from Garg (2005), the one that we use has an easier implementation while achieving satisfactory empirical results in our tests. We use $\epsilon = 10^{-3}$ in Algorithm 2.1. After solving the problem, we perform local search, including 2-opt and relocation (Toth and Vigo, 2014), to improve the solutions. The relocation procedure used in the local

search works as follows. After we finish the 2-opt, we attempt to reassign the nodes in the most lengthy tour to another compatible vehicle and insert them to the best position of the tour. We repeat this process until no improvements can be made.

We code our algorithm in Java and execute the tests on a computer with an Intel Core i7-3770 CPU running at 3.4 GHz and 8 GB of RAM. We use Gurobi 7.5.1 as the mixed integer programming solver. We report the results with 10 minutes and 2 hours (120 minutes) time limits for MIP solver.

We evaluate the proposed algorithm against the MIP on small instances with up to 26 nodes and large instances with up to 101 nodes. For each test instance, we report the lower bounds, LB_1 (LB_2), and upper bounds, UB_1 (UB_2), found by MIP within the 10 minutes (2 hours) time limit, the CPU time for MIP in seconds (Time) if the problem is solved within the time limit, the objective value of the solution found by approximation algorithm (Obj), the CPU time for the approximation algorithm in seconds (Time), the empirical approximation ratio computed as $\frac{\text{Obj}}{\text{LB}_2}$, and the ratio between the objective from the approximation algorithm and the best upper bound from MIP, $\frac{\text{Obj}}{\text{UB}_2}$. In the second MIP run (2 hours limit), we highlight (with bold font) those lower/upper bounds that were improved from the first MIP run (10 minutes limit).

Tables 2.1 and 2.2 summarize the results for instances with up to 26 nodes and different compatibility constraints. The instances with relaxed compatibility constraints are more difficult to solve. Solving VRPCC through MIP is challenging for the state-of-art solver. Out of twelve instances each, two and six instances cannot be solved to optimality within the time limit for tight instances and relaxed instances, respectively. Our proposed algorithm works well for these small instances as most of them can be solved within one second. Comparing the best lower bound found by MIP, our proposed algorithm yields good empirical approximation ratios: the ratios are within 1.16 for all instances with tight compatibility constraints and within 2 for ten out of twelve instances with relaxed compatibility constraints.

Table 2.1: Numerical results for instances with up to 26 nodes and tight compatibility constraints

Instance	MIP (10 minutes)			MIP (2 hours)			Approximation			
	LB_1	UB_1	Time (s)	LB_2	UB_2	Time (s)	Obj	Time (s)	$\frac{Obj}{LB_2}$	$\frac{Obj}{UB_2}$
C-n11-k4	49.30	49.30	0.01	49.30	49.30	0.01	49.30	0.07	1.00	1.00
C-n16-k4	81.40	81.40	1.22	81.40	81.40	1.27	83.30	0.19	1.02	1.02
C-n21-k6	81.60	81.60	62.56	81.60	81.60	58.62	87.70	0.18	1.07	1.07
C-n26-k6	77.10	86.10	600.01*	**78.30**	86.10	7200.08*	86.20	0.36	1.10	1.00
R-n11-k4	97.00	97.00	0.09	97.00	97.00	0.09	98.10	0.04	1.01	1.01
R-n16-k4	108.90	108.90	0.10	108.90	108.90	0.11	108.90	0.05	1.00	1.00
R-n21-k6	96.80	96.80	1.56	96.80	96.80	1.36	96.80	0.11	1.00	1.00
R-n26-k6	108.90	108.90	2.14	108.90	108.90	2.02	126.60	0.20	1.16	1.16
RC-n11-k4	89.40	89.40	0.06	89.40	89.40	0.06	95.80	0.02	1.07	1.07
RC-n16-k4	131.00	131.00	0.42	131.00	131.00	0.36	131.30	0.05	1.00	1.00
RC-n21-k6	156.60	156.60	0.13	156.60	156.60	0.12	156.60	0.07	1.00	1.00
RC-n26-k6	130.50	131.10	600.01*	**130.80**	131.10	7200.02*	138.80	0.27	1.06	1.06

*: computation reaches time limit

Table 2.2: Numerical results for instances with up to 26 nodes and relaxed compatibility constraints

Instance	MIP (10 minutes)			MIP (2 hours)			Approximation			
	LB_1	UB_1	Time (s)	LB_2	UB_2	Time (s)	Obj	Time (s)	$\frac{Obj}{LB_2}$	$\frac{Obj}{UB_2}$
C-n11-k4	41.20	41.20	1.39	41.20	41.20	1.33	41.70	0.12	1.01	1.01
C-n16-k4	47.18	79.00	600.10*	**50.60**	**78.60**	7200.21*	80.30	0.25	1.59	1.02
C-n21-k6	25.89	83.60	600.12*	**39.65**	**81.20**	7200.47*	81.30	1.13	2.05	1.00
C-n26-k6	21.54	83.30	600.05*	**27.18**	**81.60**	7200.35*	81.30	1.57	2.99	1.00
R-n11-k4	76.80	76.80	1.21	76.80	76.80	1.65	87.40	0.04	1.14	1.14
R-n16-k4	89.90	89.90	33.32	89.90	89.90	33.29	98.20	0.32	1.09	1.09
R-n21-k6	61.50	83.10	600.01*	**70.00**	**80.00**	7200.07*	99.50	0.29	1.42	1.24
R-n26-k6	72.57	107.10	600.02*	**79.03**	**93.30**	7200.14*	115.60	0.30	1.46	1.24
RC-n11-k4	81.30	81.30	3.86	81.30	81.30	4.52	88.30	0.02	1.09	1.09
RC-n16-k4	80.64	89.10	600.02*	**89.10**	89.10	1632.83	116.40	0.10	1.31	1.31
RC-n21-k6	90.20	90.20	152.56	90.20	90.20	155.92	90.40	0.35	1.00	1.00
RC-n26-k6	33.60	96.30	600.03*	**49.48**	**96.10**	7200.22*	95.20	0.51	1.92	0.99

*: computation reaches time limit

Tables 2.3 and 2.4 summarize the results for instances with up to 101 nodes and different compatibility constraints. It becomes very difficult to obtain a good lower bound by using the state-of-art solver for MIP, even given a two-hour time limit. Regarding the runtime, our proposed algorithm is capable of solving two types of instances with 101 nodes in 15 seconds and 270 seconds, respectively. The empirical approximation ratios are

Table 2.3: Numerical results for instances with up to 101 nodes and tight compatibility constraints

Instance	MIP (10 minutes)			MIP (2 hours)			Approximation			
	LB_1	UB_1	Time (s)	LB_2	UB_2	Time (s)	Obj	Time (s)	$\frac{Obj}{LB_2}$	$\frac{Obj}{UB_2}$
C-n21-k6	87.30	87.30	5.79	87.30	87.30	6.47	87.30	0.01	1.00	1.00
C-n41-k10	109.70	114.30	600.03*	**114.30**	114.30	1772.48	114.50	0.09	1.00	1.00
C-n61-k14	55.07	–	600.15*	**55.35**	**102.70**	7200.15*	102.90	1.50	1.86	1.00
C-n81-k18	72.69	–	600.02*	**72.79**	**125.50**	7200.25*	117.40	2.82	1.61	0.94
C-n101-k22	32.13	–	600.02*	**32.39**	–	7200.22*	122.70	15.44	3.79	–
R-n21-k6	120.70	120.70	3.23	120.70	120.70	4.10	124.90	0.02	1.03	1.03
R-n41-k10	100.70	103.20	600.05*	101.90	**103.00**	7200.05*	149.70	0.28	1.47	1.45
R-n61-k14	77.80	121.10	600.04*	**81.20**	**120.10**	7200.08*	126.80	1.03	1.56	1.06
R-n81-k18	54.02	–	600.04*	**54.88**	–	7200.52*	117.60	3.73	2.14	–
R-n101-k22	51.80	–	600.05*	**56.40**	–	7200.26*	121.30	11.27	2.15	–
RC-n21-k6	138.80	138.80	0.80	138.80	138.80	1.01	138.80	0.03	1.00	1.00
RC-n41-k10	194.90	194.90	277.41	194.90	194.90	295.56	194.90	0.33	1.00	1.00
RC-n61-k14	108.40	257.30	600.03*	**114.00**	**146.30**	7200.11*	170.50	2.09	1.50	1.17
RC-n81-k18	49.81	–	600.03*	**50.34**	–	7200.17*	170.60	4.67	3.39	–
RC-n101-k22	47.08	–	600.04*	**47.61**	–	7200.97*	178.50	9.32	3.75	–

*: computation reaches time limit; –: no upper bounds found by MIP within time limit

Table 2.4: Numerical results for instances with up to 101 nodes and relaxed compatibility constraints

Instance	MIP (10 minutes)			MIP (2 hours)			Approximation			
	LB_1	UB_1	Time (s)	LB_2	UB_2	Time (s)	Obj	Time (s)	$\frac{Obj}{LB_2}$	$\frac{Obj}{UB_2}$
C-n21-k6	29.85	82.80	600.10*	**41.85**	**81.60**	7200.38*	80.60	0.06	1.93	0.99
C-n41-k10	18.25	129.20	600.08*	**18.44**	**86.50**	7200.18*	89.00	3.70	4.83	1.03
C-n61-k14	15.06	–	600.30*	**15.43**	–	7200.09*	90.40	32.31	5.86	–
C-n81-k18	13.36	–	600.07*	**15.48**	–	7200.39*	117.00	100.65	7.56	–
C-n101-k22	14.00	–	600.47*	14.00	–	7200.25*	117.00	262.38	8.35	–
R-n21-k6	56.58	79.90	600.03*	**65.09**	79.90	7200.05*	82.60	0.29	1.27	1.03
R-n41-k10	42.22	181.60	600.07*	**43.14**	**107.80**	7200.16*	97.80	3.08	2.27	0.91
R-n61-k14	35.83	–	600.24*	**36.48**	–	7200.09*	94.30	24.85	2.58	–
R-n81-k18	29.09	–	600.10*	**32.53**	–	7200.11*	103.90	89.01	3.19	–
R-n101-k22	26.18	–	600.19*	26.18	–	7200.05*	99.80	231.50	3.81	–
RC-n21-k6	90.20	90.20	141.53	90.20	90.20	142.03	90.40	0.27	1.00	1.00
RC-n41-k10	17.77	228.50	600.11*	**19.89**	**125.20**	7200.18*	160.50	4.41	8.07	1.28
RC-n61-k14	26.44	–	600.09*	**26.82**	–	7200.19*	117.50	31.39	4.38	–
RC-n81-k18	24.93	–	600.07*	**29.23**	–	7200.22*	121.00	114.88	4.14	–
RC-n101-k22	23.91	–	600.15*	23.91	–	7200.40*	124.20	268.86	5.19	–

*: computation reaches time limit; –: no upper bounds found by MIP within time limit

maintained within 8; we think that this is a pessimistic bound as the lower bounds from MIP seem poor. In addition, our results are comparable to the best MIP solutions found within the time limit as ratios $\frac{Obj}{UB_2}$ are close to 1; in fact, in many cases the approximation

algorithm finds a better solution.

2.5 Concluding Remarks

We studied a VRP with minimum makespan objective and compatibility constraints. This problem has a variety of potential applications, including shared mobility. We obtained a $(2\ln n+1)$-approximation algorithm using a set covering framework, and proved a nearly matching $(\ln n)$-hardness of approximation. Our numerical experiments showed that the VRPCC is challenging to solve directly using integer programming and the proposed algorithm is efficient and yields an empirical approximation ratio that is much better than its theoretical bound.

CHAPTER 3

Improving Column Generation for Vehicle Routing Problems via Random Coloring and Parallelization

3.1 Introductory Remarks

We consider the VRP with unit demand (VRPUD) that can be used to model a variety of service operations management problems in practice, where we route a fleet of vehicles to visit a set of customers, and each vehicle has a capacity, i.e., an upper bound on the maximum number of customers to visit. We describe the problem setup mathematically as follows. Let $G = (V \cup \{s\}, E)$ be an undirected graph, where $V = \{1, 2, \ldots, n\}$ is the set of nodes representing customer locations and s is the depot node. The set $E = \{(i, j)|\ i, j \in V \cup \{s\}\}$ contains all the edges representing best travel routes between each pair of customers. Each edge $(i, j) \in E$ is associated with a deterministic travel cost $c_{ij} > 0$, which satisfies the triangle inequality, that is $c_{ij} + c_{jp} \geq c_{ip}$ for all edges $(i, j), (j, p), (i, p) \in E$. A fleet of identical vehicles located at the depot node, denoted by set K, is used to serve all the customer nodes in V, and each vehicle has capacity Q. We assume that a unit demand is attached to each node $i \in V$. Our goal is to find a set of vehicle routes with the minimum total travel cost such that: (i) each node in V is visited by exactly one route, (ii) all vehicles start and end their routes at the depot node s, and (iii)

each route contains at most Q customer nodes.

VRPUD finds natural applications in service systems in transportation, logistics, and healthcare. One example application is routing in a patient-centered medical home system, where we need to route caregivers to provide medical care services at patients' homes (American Academy of Family Physicians, 2008). The patient-centered medical home system benefits patients with reduced mobility and decreases the number of hospital admissions (Adaji et al., 2018; The National Association for Home Care & Hospice, 2010). However, the routing and scheduling tasks are highly complex and often done manually and therefore result in high operational costs (Eveborn et al., 2006). According to The National Association for Home Care & Hospice (2010), nationwide, the number of patients visited by one caregiver ranges from 4 to 6 during a workday, depending on the types of caregivers. Using this information, we can formulate the problem as a VRPUD to find the optimal routing strategy of the caregivers by specifying the capacity Q of each vehicle. Fikar and Hirsch (2017) review different routing approaches under the context of patient-centered medical home, e.g., modeling the routing and scheduling problem as VRP with time windows (VRPTW) or VRP with pickup and delivery. However, the approach of solving the problem as VRPUD, considering the workload of each caregiver, has not been studied.

Note that when we eliminate the capacity requirement, VRPUD extends the classic traveling salesman problem, which finds a least-cost route that visits all nodes in a given network (Kruskal, 1956). VRPUD is also a special case of the capacitated VRP (CVRP) where a fleet of vehicles, each having a limited capacity, is routed to visit a set of customers with different demand requirements (see, e.g., Baldacci et al., 2011; Fukasawa et al., 2006; Pecin et al., 2017b; Toth and Vigo, 2014).

3.1.1 Column Generation Overview

Column generation is a prominent approach to solving VPRs (see, e.g., Baldacci et al., 2010, 2008, 2011; Desaulniers et al., 2006; Fukasawa et al., 2006; Pecin et al., 2017a,b). In column generation, we apply a set-partitioning formulation for the VRP where we associate a binary decision variable to each feasible route. A key first step is to solve the linear program (LP) relaxation of this integer program. Because the number of decision variables in the "master" LP can be huge, a restricted LP containing only a subset of all possible columns (i.e., decision variables) is repeatedly solved. An optimal solution to this LP provides a dual solution, based on which we can either improve the solution to the master LP or verify optimality of the current solution. This requires solving the so-called "pricing problem" which finds a new column with the minimum reduced cost. If the minimum reduced cost is non-negative, then the current solution is optimal for the master LP. Otherwise, the column with the least reduced cost enters the basis and we re-optimize the restricted LP with an expanded set of columns. We note that in each iteration, any column with negative reduced cost could improve the solution. Therefore, we are not restricted to only add columns with the least reduced cost, and can add any columns with negative reduced cost (Desaulniers et al., 2006).

The combinatorial algorithms to solve the pricing problem in column generation play an essential role in improving computational efficiency. Finding a column with negative reduced cost is equivalent to solving an elementary shortest path problem with resource constraints (ESPPRC), which finds the minimum cost path starting and ending at the depot, traversing customer nodes under required resource constraints. When the underlying network contains negative cost arcs, the problem becomes NP-hard (Dror, 1994). Desrosiers et al. (1995) proposed a dynamic programming method to solve a relaxed version of ESPPRC that allows cycles. Based on their work, Feillet et al. (2004) proposed a label correcting algorithm that is the first exact solution method for ESPPRC, and it was later improved by Feillet et al. (2007). This algorithm could solve the pricing problem of the VRPTW very

efficiently when time windows are tight, but it failed to handle problem instances with wide time windows. Later, Righini and Salani (2006) proposed a bi-directional label correcting algorithm that relies on state-space relaxation for ESPPRC and significantly improved the computational efficiency. Recently, Lozano et al. (2015) proposed an algorithm (called pulse) that efficiently solved ESPPRC arising in VRPTW using implicit enumeration and a bounding procedure. The algorithm worked with a depth-first-search-based enumeration to construct the partial paths starting from the depot node to the end node. With several pruning strategies to discard the search on the partial paths early, the algorithm prunes large regions of the solution space and efficiently solves the problem. The algorithm was later extended and generalized by Duque et al. (2015) as a general-purpose framework for hard shortest path problems and for the orienteering problem with time windows.

The pricing problem in column generation for VRPUD has a parameter Q which is the maximum number of nodes in any solution route. Although the ESPPRC is NP-hard, it is possible to find a polynomial-time algorithm with respect to the size of the input when the parameter Q is fixed. Fixed parameter tractability is a branch in computational complexity theory that focuses on classifying computational problems with respect to the parameters of the input (Downey and Fellows, 2012). It separates the complexity of an NP-hard problem into two parts: one depends on the size of the input, and the other depends on some fixed parameter. If the parameter of the instance is small, then we might find a polynomial-time solution to that instance. In our context, there are polynomial-time algorithms for finding a simple path/cycle of a fixed length based on a technique called color-coding introduced by Alon et al. (1995). In the color-coding method, each node is randomly assigned a color from a set with a fixed number of colors. Finding a simple path of a fixed length then reduces to finding a path containing nodes with distinct colors. The complexity of the latter problem is exponential with respect to the number of colors but polynomial with respect to the number of nodes; this is because we only need to track the subset of previously-visited colors rather than nodes. However, as one color-coding could color two distinct nodes with

the same color and forbid the exploration of the path containing the two nodes, we need to investigate multiple independent trials of color-coding. In this chapter, we utilize this approach to obtain an efficient algorithm for ESPPRC instances in VRPUD with a fixed capacity of Q.

In the column generation method, at the end, the solution to the master LP may not be integral. One common approach is to use all the generated columns to solve the restricted problem as an integer program in order to obtain a heuristic integral solution, along with an optimization gap between the LP and integer program solutions. To solve the VRP exactly, additional techniques are needed: branch-and-price (Barnhart et al., 1998) is used to close the optimization gap by applying a branch-and-bound framework atop the column generation, and branch-cut-and-price (Fukasawa et al., 2006) improves the branch-and-price method by introducing cutting planes to strengthen the lower bound on each branching node.

In this chapter, we focus on solving the LP relaxation exactly using new algorithms for the pricing problem. We also use the generated columns to find a heuristic integer solution.

3.1.2 Contributions

The main contributions of this chapter are threefold.

First, we demonstrate the use of the "random coloring" idea as a pricing algorithm (to solve ESPPRC) in the column generation method for VRPs. In particular, we consider the VRPUD problem, and obtain an efficient algorithm to solve its column-generation-based LP relaxation. We believe that this approach is applicable more broadly to other VRPs with some (possibly implicit) limit on the number of nodes visited in each route. As different coloring schemes are completely independent, it is natural to implement this algorithm in parallel, and we observe a high speed-up ratio in our parallel implementation. We note that techniques to accelerate classical label-correcting algorithms for ESPPRC, such as bi-directional search and bounding functions, are also applicable to our algorithm, and may

yield further speed-up.

Second, we evaluate the random coloring approach against a state-of-the-art algorithm for ESPPRC, the pulse algorithm (discussed earlier). The pulse algorithm has previously been tested in the column generation approach for VRPTW instances, but it relies on a bounding scheme that only considers time consumption. In this chapter, we extend the pulse algorithm to VRPUD by (1) extending the bounding scheme to consider the capacity consumption of the vehicle, and (2) allowing the algorithm to stop early and return multiple negative-cost paths instead of just one optimal path. The comparison between the pulse algorithm and the random coloring algorithm has been tested on modified Solomon's instances (Solomon, 1987) and unitary demand CVRP X-instances (Uchoa et al., 2017) with the size of the instances ranging from 50 nodes to 957 nodes. Each route is set to contain at most 6 nodes. Computational results show that the parallel implementations of both algorithms can solve the LP relaxations of these instances efficiently, and the heuristic integer solutions found have small optimality gaps. The optimality gap is in general within 5% for both pricing algorithms. The random coloring algorithm is more efficient than the pulse algorithm for instances with smaller vehicle capacity.

Third, we extend our work to the multi-depot VRPUD and benchmark the proposed algorithms on patient-centered medical home instances based on census data in Wayne County, Michigan. Both the pulse and random coloring algorithms can solve the LP relaxation of VRPUD instances with up to 500 nodes (and $Q = 4$) in a reasonable time in parallel: about 8 minutes for random coloring and 15 minutes for pulse. Moreover, using the generated columns in the integer program, we can obtain high-quality integer solution. For census data instances, the optimality gap is even better comparing to the VRPUD benchmark instances: it is generally within 2% via both algorithms.

3.1.3 Other Related Work

As discussed before, the column generation method only applies to the LP relaxation, and to obtain an exact solution one needs to embed this method within the branch-and-bound or branch-and-cut framework. In recent years, the branch-cut-and-price approach, which combines column generation with branch-and-cut framework, has been considered the most efficient exact solution approach for VRPs and has been widely applied to solve various variants of VRPs (Pecin et al., 2017a,b). Nevertheless, solving the LP relaxation efficiently via column generation is an important aspect of the overall exact approach for VRPs.

The branch-and-price method performs very well for VRPTW. Desrosiers et al. (1995) consider a column generation method for VRPTW that models the pricing problem as a shortest path problem with resource constraints that allows cycles, which is later improved by Kohl et al. (1999) and Irnich and Villeneuve (2006) forbidding cycles with a fixed size. Feillet et al. (2004) first solve VRPTW using column generation with ESPPRC as the pricing problem, which is harder to solve as the path is not allowed to repeat nodes. They propose the first exact method for the ESPPRC to improve the lower bounds obtained at each branching node. Jepsen et al. (2008) extend the branch-cut-and-price framework by introducing so-called "subset-row" cuts which effectively enhances the bound from root node relaxation. Baldacci et al. (2010) propose a column-and-cut generation method and use non-elementary route relaxation approach to bound the pricing problem. Pecin et al. (2017a) propose a branch-cut-and-price approach that combines several recently developed algorithms with limited-memory subset-row cuts and improved elementary inequalities.

CVRP can also be viewed as a special case of VRPTW with arbitrary large time windows. However, with resource (time) being less constrained, the pricing problem formulated as ESPPRC becomes more challenging to solve. To solve CVRP instances through column-generation-based approaches, the main stream of the research focuses on considering non-elementary relaxations of the pricing problem and strengthening the formulation through various cutting planes. Christofides et al. (1981) first introduce q-route relaxation,

which is a walk with at most q units of load that starts at the depot, traverses a sequence of nodes and then returns to the depot. In a q-route, we allow a vehicle to visit the same node multiple times, which may create loops. It is easy to avoid 2-node loops but it is hard to avoid k-node loops with $k \geq 3$. Fukasawa et al. (2006) initiate a branch-cut-and-price method to solve CVRP by combining branch-and-cut and column generation. They consider the pricing problem as a minimum cost q-route problem without 2-node loops, which significantly reduces the solution time of the pricing problem. Fukasawa et al. (2015) extend the algorithm to solve a variant of CVRP where the cost of an arc is defined as the product of the arc length and the load of a vehicle traveling on this arc.

A variety of column-generation studies for CVRP have focused on finding columns associated with elementary routes, whose efficiency relies on bounding functions to reduce the search space of a dynamic program. The bounds are computed through different state-space relaxations. Baldacci et al. (2008) proposed a column-and-cut generation approach using a bounding procedure combining three dual ascent heuristics. Baldacci et al. (2011) introduced the concept of ng-route that is more effective than the q-route. The ng-route is a non-elementary route limiting the visit to a node that was previously visited if such a node belongs to a dynamically computed no-good (ng) set associated with the route. Adding another dual ascent heuristic with ng-route relaxation and non-robust subset-row cuts, they improved the speed and stability of the branch-cut-and-price algorithm.Recently, Pecin et al. (2017b) improved the branch-cut-and-price algorithm by incorporating and enhancing various techniques from the past 10 years.

Table 3.1 summarizes the reviewed literature applying column generation-based methods for VRPs. We classify them based on solution approaches used for solving the pricing problem. We refer the interested readers to a survey paper by Braekers et al. (2016) for state-of-the-art classification and theory development for vehicle routing-related problems.

We also note that the color-coding approach has been used computationally in a different setting, namely in bioinformatics to explore complex structures in protein-protein

Table 3.1: Summary of the reviewed papers based on solution methods for the pricing problem

Class of VRP	ESPPRC	Non-elementary Route Relaxation
VRPTW	Feillet et al. (2004), Feillet et al. (2007), Jepsen et al. (2008), Righini and Salani (2006), Lozano et al. (2015), Pecin et al. (2017a), etc.	Desrosiers et al. (1995), Kohl et al. (1999), Irnich and Villeneuve (2006), Baldacci et al. (2010), etc.
CVRP	Baldacci et al. (2008), Baldacci et al. (2011), Pecin et al. (2017b), etc.	Fukasawa et al. (2006), Fukasawa et al. (2015), Baldacci et al. (2008), Baldacci et al. (2011), Pecin et al. (2017b), etc.
Other VRPs	Bettinelli et al. (2011), Dabia et al. (2013), Duque et al. (2015), etc.	Dabia et al. (2013), etc.

interaction networks (Alon et al., 2008).

3.1.4 Outline

The remainder of the chapter is organized as follows. In Section 3.2, we define the VRPUD and provide a set-partitioning-based formulation that can be solved via column generation. In Section 3.3, we discuss two parallel algorithms that efficiently solve the ESPPRC as the pricing problem in the column generation for VRPUD. The first algorithm is an extension based on the pulse framework proposed by Lozano et al. (2015) and the second algorithm is a randomized parallel algorithm applying color-coding (Alon et al., 1995). In Section 3.4, we present the performance results of the proposed approaches on (i) modified classic VRP benchmark instances and (ii) large-size patient-centered medical home delivery instances.

3.2 VRPUD and Column Generation

Consider the VRP with unit demand (VRPUD) as defined in Section 3.1. A set-partitioning-based formulation of our problem is given as follows. Let P be the set of feasible routes, each containing at most Q nodes, which can be assigned to each vehicle in K. For each feasible route $p \in P$ and node $i \in V$, let c_p be the cost of the route and a_{ip} be the binary coefficient such that $a_{ip} = 1$ if route p contains i and $a_{ip} = 0$ otherwise. We define binary decision variable x_p for each $p \in P$ such that $x_p = 1$ if we pick the route p in our solution and $x_p = 0$ otherwise. Then, the VRPUD is given by

$$\textbf{(MP)} \text{ minimize: } \sum_{p \in P} c_p x_p \tag{3.1}$$

$$\text{subject to: } \sum_{p \in P} a_{ip} x_p = 1 \qquad \forall\, i \in V, \tag{3.2}$$

$$x_p \in \{0, 1\} \qquad \forall\, p \in P, \tag{3.3}$$

where the objective function (3.1) minimizes the overall cost of the routing; constraints (3.2) ensure that each node in V is covered by exactly one vehicle; and lastly, constraints (3.3) enforce that all the decision variables are binary.

Model MP is hard to solve because P contains a number of feasible routes that grows exponentially with the size of the input instance. One way to address this challenge is to apply the column generation method. Instead of solving the problem with all variables explicitly, we solve a restricted master problem (RMP), where a relatively small subset of

P, $\tilde{P} \subset P$, is used to replace P. Equivalently, the RMP is given by:

$$\textbf{(RMP)} \text{ minimize: } \sum_{p \in \tilde{P}} c_p x_p \tag{3.4}$$

$$\text{subject to: } \sum_{p \in \tilde{P}} a_{ip} x_p = 1 \qquad \forall\, i \in V, \tag{3.5}$$

$$x_p \in \{0, 1\} \qquad \forall\, p \in \tilde{P}. \tag{3.6}$$

Column generation is an iterative method. In each iteration, we solve the linear relaxation of the RMP and obtain a dual solution to the problem. Using the dual solution, we can then search for new routes with negative reduced cost, which can potentially improve the objective value of the LP relaxation of RMP. In column generation for VRPUD, let $\pi_i,\ i \in V$ be the dual variables corresponding to each constraint (3.5). Then, the reduced cost of a route p (a column in the RMP) is computed as $\bar{c}_p = \sum_{(i,j) \in p} \bar{c}_{ij}$, where for each arc (i, j), $\bar{c}_{ij}$ is then calculated as $\bar{c}_{ij} = c_{ij} - \pi_j$. When we find such routes, we add them into $\tilde{P}$ and continue to the next iteration. We obtain the optimal value of the LP relaxation of MP when no more negative reduced cost routes exist. An overview of the algorithm is provided in Figure 3.1.

The main difficulty of implementing the column generation approach is at the step of finding the routes with negative reduced cost, i.e., solving the pricing problem to generate columns. In this chapter, we focus on implementing efficient algorithms to solve the pricing problem, which is formulated as an ESPPRC, described as follows. We are given a directed graph $G' = (V \cup \{s, t\}, A)$ where $V \cup \{s, t\}$ is the set of nodes with a source node s and a terminal node t, which both represent the depot node, and $A = \{(i, j) | i \in V \cup \{s\}, j \in V \cup \{t\}\}$ is the set of arcs. For each arc $(i, j) \in A$, we associate it with a cost $\bar{c}_{ij}$, which can be negative. Under the setting of VRPUD, we are also given a resource bound Q; a consumption of 1 is associated with each node $i \in V$. Our goal is to find an elementary path from source node s to terminal node t with minimum cost such that the resource constraint

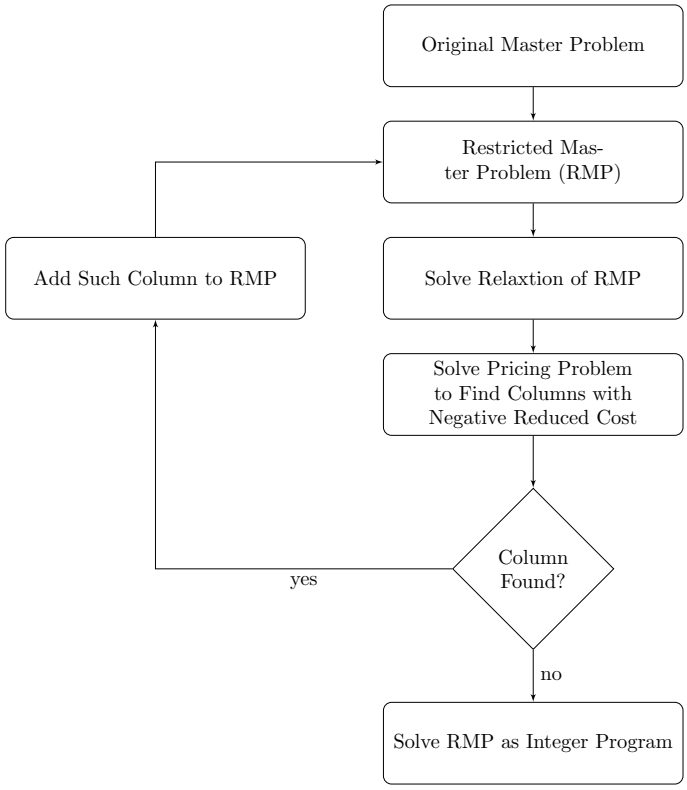


Figure 3.1: Overview of the column generation approach

is satisfied.

We define $y = (y_{ij}, (i,j) \in A)^\top$ as a binary decision variable where $y_{ij} = 1$ if we visit node j after node i in the solution path and $y_{ij} = 0$ otherwise. Let $(\pi_i : i \in V)$ denote the dual solution from the current iteration of RMP and let $\pi_s = \pi_t = 0$. The pricing problem

(ESPPRC) is equivalent to the following flow-based integer program.

$$\textbf{(PP)}\ z_{\text{PP}}(\pi) = \underset{y}{\text{minimize}} \sum_{(i,j)\in A} (c_{ij} - \pi_j) y_{ij} \tag{3.7}$$

$$\text{subject to} \sum_{j:(s,j)\in A} y_{sj} = 1 \tag{3.8}$$

$$\sum_{j:(j,t)\in A} y_{jt} = 1 \tag{3.9}$$

$$\sum_{j:(i,j)\in A} y_{ij} - \sum_{j:(j,i)\in A} y_{ji} = 0 \quad \forall i \in V \tag{3.10}$$

$$\sum_{(i,j)\in A} y_{ij} \leq Q + 1 \tag{3.11}$$

$$\sum_{(i,j)\in A,\ i,j\in S} y_{ij} \leq |S| - 1 \quad \forall S \subset V \tag{3.12}$$

$$y_{ij} \in \{0,1\} \quad \forall (i,j) \in A, \tag{3.13}$$

where the objective function (3.7) minimizes the cost of the path; constraints (3.8)–(3.9) specify the starting and ending node of the path; constraints (3.10) are flow balance constraints; constraint (3.11) ensures the path contains no more than Q nodes in V; constraints (3.12) are subtour elimination constraints that forbid any cycles in the network. After solving PP, if $z_{\text{PP}}(\pi) < 0$, then we find a path with negative reduced cost. Otherwise, our solution is optimal to the linear relaxation of RMP. Solving PP through mixed-integer programming is challenging when we need to address an exponential number of constraints (3.12).

As indicated in our literature review in Section 3.1, the usage of non-elementary route relaxation has significantly improved the computational efficiency for each branch-and-bound node but yields a worse lower bound that results in a larger branch-and-bound tree (Desaulniers et al., 2006). In this chapter, we focus on the exact pricing approach in column generation for VRPs, which is equivalent to finding the minimum cost route on a network where arcs are associated with negative costs while satisfying some side constraints such

as capacity. Contrary to tightly constrained instances where this ESPPRC can be solved efficiently (as in the pricing problem embedded in solving VRPTW), it becomes very challenging to solve for the less-constrained instances, for example, for the pricing problem embedded in solving CVRP (Lysgaard et al., 2004).

3.3 Solution Methods for the ESPPRC

Efficiently solving the pricing problem (PP) for ESPPRC is crucial to improving the performance of the column generation approach. In this section, we propose two exact algorithms for solving ESPPRC. The first algorithm (called pulse) utilizes bounding and pruning strategies to accelerate the solution speed of a dynamic program. The second algorithm (called random coloring) is a randomized algorithm that solves a dynamic program with significantly reduced state space.

3.3.1 Pulse Algorithm for ESPPRC

This algorithm is based on an idea from Lozano and Medaglia (2013) to solve the constrained shortest path problem and its extension to solve ESPPRC instances arising from VRPTW in Lozano et al. (2015). In our setting (for VRPUD), the resource consumption is the vehicle capacity used rather than time (for VRPTW).

The overall approach is to compute values $b(v, q)$ representing the minimum cost of a path from v to t that already starts with resource consumption q. These values are computed in a backward manner, starting with $q = Q$ (which is trivial) and iteratively decreasing q by a step-size Δ. In order to compute $b(v, q)$ for some q, the algorithm performs a depth-first exploration from v and uses the $b(\cdot, q+\Delta)$ values as lower bounds to prune the search (after Δ nodes have been explored). There are some other pruning strategies as well, which are described below.

In more detail, the algorithm computing $b(v, q)$ constructs paths from the starting node

v to the terminal node t by propagating from each current node to its successors. The propagation recursively explores the graph to construct partial paths while recording some information. At each node, the algorithm tries to explore all outgoing arcs unless certain pruning strategies are triggered to stop the propagation. Each time, when the propagation reaches the terminal node t, we find a feasible solution (which updates the current best solution) and the algorithm will then backtrack to explore other options. In the end, the algorithm would be able to enumerate all possible paths from s to t following a depth-first search scheme. Crucially, by implementing pruning strategies to stop exploration early, the algorithm cleverly avoids full enumeration.

The implementation of this approach uses two procedures: pulse (see Algorithm 3.1) and bound (see Algorithm 3.2). The pulse procedure takes as input a current path P, its cost $r(P)$, its load $q(P)$ and a node w to which the path is being extended. It also maintains a pair of global variables: the best path P^* found so far and its cost $r(P^*)$. The global variables are updated whenever the propagation reaches the end node t and the resulting path is better than P^*. To find more columns with negative costs per iteration in the pricing problem that is formulated as ESPPRC, we introduce a global list $\mathcal{L}$ containing paths with negative costs. We add a path to $\mathcal{L}$ whenever the propagation reaches the end node t and the resulting path has a negative cost. We terminate the algorithm early when the size of $\mathcal{L}$ reaches a preset limit, *nSol*. Note that finding the optimal path P^* is critical for the bounding procedure that will be discussed later and maintained list $\mathcal{L}$ is only used when calling pulse procedure to solve the entire problem. To efficiently explore the graph, the pulse procedure utilizes a set of pruning strategies: infeasibility, rollback, and bounds, which will be detailed in Section 3.3.1.1. The most important strategy is bounds pruning, which relies on the already-computed $b(\cdot, q+\Delta)$ values. The bound procedure implements a backward dynamic program to compute the values $b(v, q)$ for $q = Q - \Delta, Q - 2\Delta, \cdots$, each time invoking the pulse procedure. In particular, we start by obtaining (using pulse) the elementary shortest path from every node $v \in V$ to t given a resource consumption

$Q - \Delta$. Then, we continue to solve for the elementary shortest path from every node $v \in V$ to t given a resource consumption $Q - 2\Delta$. We can continue to repeat the same procedure backwards until we reach a desired lower bound on the bounding resource consumption $\underline{Q}$. Therefore, this procedure collects all $b(v, q)$ values for all $v \in V$ and $q \in \mathcal{Q}$, where $\mathcal{Q} = \{\underline{Q}, \underline{Q} + \Delta, \ldots, Q - 2\Delta, Q - \Delta\}$.

Algorithm 3.1: Pulse procedure

input : Current node w; cost $r(P)$; path load $q(P)$; current path P
output: Void

Let u and v be the second last and the last node visited in P, respectively
if $w == t$ **then**
  if $r(P) + c'_{vw} < r(P^*)$ **then**
    $P^* \leftarrow P \cup \{t\}$
    $r(P^*) \leftarrow r(P) + c'_{vw}$
  end
  ▷ update optimal path **if** $r(P) + c'_{vw} < 0$ ▷ *skip when executed inside bounding procedure* **then**
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{P \cup \{t\}\}$
  end
  stop
end
if $|\mathcal{L}| \geq nSol$ **then** stop ▷ skip when executed inside bounding procedure

if $q(P) == Q$ *or* $w \in P$ **then** stop ▷ pruned by infeasibility

if $|P| \geq 2$ *and* $c'_{uv} + c'_{vw} > c'_{uw}$ **then** stop ▷ pruned by rollback

let $\underline{q(P)}$ be the greatest q such that $q \leq q(P)$ and $q \in \mathcal{Q}$ **if** $r(P) + b(w, \underline{q(P)}) \geq r(P^*)$ **then** stop ▷ pruned by bounds

$P' \leftarrow P \cup \{w\}$
$q(P') \leftarrow q(P) + 1$
$r(P') \leftarrow r(P) + c'_{vw}$ ▷ $r(P') \leftarrow 0$ if $P = \emptyset$
for $(w, w') \in A$ **do**
  $pulse(w', r(P'), q(P'), P')$
end

The overall algorithm works as follows. We start by executing the bounding procedure to compute the lower bound matrix B. Note that we do not maintain the list of negative

Algorithm 3.2: Bounding procedure

input : Graph $G' = (V \cup \{s, t\}, A)$; step size Δ; bounding cap $[\underline{Q}, Q]$
output: Lower bound matrix $B = [b(v, q) : v \in V,\ q \in \mathcal{Q}]$

$q \leftarrow Q$ **while** $q > \underline{Q} + \Delta$ **do**
  $q \leftarrow q - \Delta$ **for** $v \in V$ **do**
    $P^* \leftarrow \{\}$ ▷ initialize global variables
    $r(P^*) \leftarrow \infty$
    $P \leftarrow \{\}$
    $r(P) \leftarrow 0$
    $q(P) \leftarrow q$
    $pulse(v, r(P), q(P), P)$ ▷ find the optimal partial path from v to t given q consumed
    $b(v, q) \leftarrow r(P^*)$
  end
end
return B

cost paths $\mathcal{L}$ when executing pulse within the bounding procedure. Next, we run the pulse procedure with $P = \{s\}$, $r(P) = 0$, and $q(P) = 0$. When the program terminates, the global list $\mathcal{L}$ contains at most *nSol* many s–t paths with negative costs.

3.3.1.1 Pruning Strategy

The efficiency of the pulse algorithm depends on the pruning strategies to stop the exploration of partial paths as soon as possible. Lozano et al. (2015) propose three pruning strategies: infeasibility, bound and rollback. Based on our problem setting, we detail how to modify each pruning strategy as follows.

Infeasibility pruning. Infeasibility pruning terminates an exploration when a partial path violates any feasibility constraints: the partial path visits more than Q nodes, or the partial path forms a cycle when it reaches a new node. For each partial path, we maintain an indicator vector of length $|V|$ to indicate if such a path has visited each node $v \in V$. We can then identify if any cycle is created in constant time, i.e., if the path extended to a node that has been previously visited.

Bounds pruning. Bounds pruning is the key component of the pruning process that

significantly improves the performance of the algorithm. The idea is to fathom suboptimal partial paths using the continuously updated primal bound $r(P^*)$ (the cost from the current best feasible solution) and pre-calculated conditional lower bounds $b(v, \underline{q(P)})$, which store the minimum reduced cost that can be achieved for every node $v \in V$ and for a given resource consumption $\underline{q(P)}$. We terminate the exploration for a partial path P when it reaches a node $v \in V$ where its cost, $r(P)$, plus the conditional lower bound at v with $\underline{q(P)}$ resource consumption is at least the current primal bound, i.e., $r(P) + b(v, \underline{q(P)}) \geq r(P^*)$. Note that we may not have a valid s–t path of cost $r(P) + b(v, \underline{q(P)})$, but it is still a lower bound.

Rollback pruning As the pulse algorithm implicitly enumerates the search space in a depth-first search fashion, a poor decision made at early stages may lead to an unpromising region of the search space. To avoid this behavior, we impose the rollback pruning strategy that examines the last choice made. Let P_{ij} be a partial path with end node j that visited node i right before j. When we extend P_{ij} to next node v, we check if $\bar{c}_{ij} + \bar{c}_{jv} > \bar{c}_{iv}$. If yes, we terminate the current exploration as a better propagation is to roll back to the partial path with end node i and extending it to v ("Rollback" is automatically done when we propagate the path from node i); otherwise, we continue the exploration. This helps to avoid bad early explorations.

3.3.1.2 Parallelization

In the pulse framework, Algorithm 3.1 explores partial paths in a depth-first search fashion. Along the search, it runs the pulse procedure on one node at a time until the search reaches the end node. Starting at the node s, the extensions starting on the different out-going arc are independent, and therefore we can implement Algorithm 3.1 in parallel on different computer threads to accelerate the search while maintaining the global information properly. Lozano and Medaglia (2013) propose to trigger a fixed number of threads at node s and explore the extensions on different out-going arcs from s independently, and we only

need to maintain the record of the visited nodes for each thread and the bound information globally. Multiple threads can run Algorithm 3.1 on the same node at the same time except for the end node t, where the global lower bound can only be updated by one thread at a time.

3.3.2 Random Coloring Algorithm for ESPPRC

A traditional way to solve ESPPRC is through the label correcting algorithm (e.g., Feillet et al., 2004; Lysgaard et al., 2004). However, to make sure the path is elementary, the algorithm needs to record the full path for each state variable. Therefore, it requires exponentially many state variables. To be specific, the size of the state space for label correcting algorithm is in the order of $O(2^{|V|}|V|)$. In this section, we discuss how to utilize the idea of color-coding from Alon et al. (1995) to extend the label correcting algorithm and efficiently cut the size of the state space to $O(2^Q|V|)$.

In VRPUD, each route can visit at most Q nodes. Suppose that we are given a color-coding, which is a function $\phi : V \to \{1, 2, \ldots, Q\}$ that maps each node in V to a color attribute labeled from $1, 2, \ldots, Q$. We say that a path in G' is *colorful* if the nodes in the path are colored by distinct colors. Clearly, every colorful path is elementary, and each colorful path contains no more than Q nodes in V. Then if we can find a colorful s–t path with negative cost, we find an elementary path connecting nodes s and t with a negative cost. To find a colorful path in G', we can modify the label correcting algorithm from Feillet et al. (2004).

Let P_{si} be a partial path from source node s to node $i \in V$. Different from the original algorithm, we record the information of color history instead of node history of the path. A state $R_i = (n_i, V_i^1, \ldots, V_i^Q)$ corresponds to the number of visited nodes and a binary indication vector that is used to record color usage, where $V_i^k = 1$ if P_{si} visits a node colored $k \in \{1, 2, \ldots, Q\}$ and $V_i^k = 0$ otherwise. Let $C_i = c(P_{si})$ be the cost of such path. A dominance rule is enforced to eliminate additional paths P_{si} in the label correcting

algorithm. Let P'_{si} and P^*_{si} be two distinct paths from s to i with associated labels (R'_i, C'_i) and (R^*_i, C^*_i). We say that P'_{si} dominates P^*_{si} if and only if $C'_i \leq C^*_i$, $n'_i \leq n^*_i$, $V'^k_i \leq V^{*k}_i$ for all $k \in \{1, 2, \ldots, Q\}$, and $(R'_i, C'_i) \neq (R^*_i, C^*_i)$. Note that the number of possible states R_i is at most $|V| \cdot 2^Q$.

The label correcting algorithm works as follow. For each node $i \in V$, we maintain a list Λ_i of paths from source node s to node i. We start with a set of active nodes containing s only. In each iteration, we poll an active node i from the active node set and extend the paths in Λ_i. Let P_{sj} be the extended path that is feasible. Suppose P_{sj} is not dominated by other paths in Λ_j; then we put j into the active node set and iterate the previous procedure. We stop the algorithm when no active nodes exist. The details of the label correcting algorithm are displayed in Algorithm 3.3. For any partial path P_{si}, we record the history of colors instead of nodes: during the extension process, we can extend a path to a new node only if we have not visited a node with the same color before.

Theorem 3.1 (Theorem 3.4 from Alon et al. (1995))**.** Let $G' = (V \cup \{s, t\}, A)$ be a directed graph. Any pairs of vertices connected by a path with Q vertices in G can be found in $O(2^Q|V||A|)$ worst-case time.

Recall that our pricing problem is defined on a network $G' = (V \cup \{s, t\}, A)$ and it suffices to output any route with negative cost. Hence, we can terminate the algorithm early to output such a solution.

Note that any negative-cost colorful path found by Algorithm 3.3 is indeed an elementary path with negative reduced cost. On the other hand, Algorithm 3.3 may fail to find a negative-cost colorful path even if there is some elementary path with negative reduced cost. We now bound this "failure" probability. For a randomly chosen coloring ϕ, any elementary path in G' with at most Q nodes (in particular, any feasible path with negative reduced cost) has a probability $\frac{Q!}{Q^Q} > e^{-Q}$ to be colorful. So the probability that the algorithm fails to identify a negative cost elementary path with at most Q nodes is less than $1 - e^{-Q}$. Then, if we repeat k independent runs of the color-coding algorithm, the prob-

Algorithm 3.3: Algorithm for ESPPRC with Colors

input : Graph $G' = (V \cup \{s, t\}, A)$, color-coding ϕ
output: A set of routes with negative costs T

Initialization $\Lambda_s \leftarrow \{(0, \ldots, 0)\}$
for $i \in V \cup \{t\}$ **do**
  $\Lambda_i \leftarrow \emptyset$
end
$S = \{s\}$
while $S \neq \emptyset$ **do**
  Pick $i \in S$
  if $i == t$ **then**
    add corresponding routes from Λ_t with negative cost to T
  end
  else
    forall $j : (i, j) \in E$ **do**
      forall $\lambda_i = (R_i, C_i) \in \Lambda_i$ **do**
        if $V_i^{\phi(j)} = 0$ **then**
          extend λ_i to get λ_j
          if λ_j *is not dominated by any path in* Λ_j **then**
            add λ_j to Λ_j and $S = S \cup \{j\}$
            remove any path in Λ_j that is dominated by λ_j
          end
        end
      end
    end
  end
  remove i from S
end
return T

ability of failing to identify a negative cost path in all repetitions is at most $(1 - e^{-Q})^k$, which is decreasing exponentially in k. Therefore, we repeat this algorithm multiple times to increase the probability of finding a colorful path with negative cost. For example, with $Q = 4$ and $k = 40$, the probability of failure is at most 0.02.

Our overall algorithm works as follows. We pre-define a stopping criterion in terms of the maximum number of iterations and a threshold count for the number of output routes. In each iteration, we randomly generate a color-coding ϕ that assigns color labels to each node in G'. Then, based on the color-coding ϕ, we solve the ESPPRC through Algorithm 3.3 and store all solution routes found with negative cost. If we reach the maximum number of iterations or the set of solutions contains more than the threshold number of output routes, we stop the algorithm; otherwise, we move to the next iteration. The detail of our random coloring algorithm for ESPPRC is presented in Algorithm 3.4.

Algorithm 3.4: Random Coloring Algorithm for ESPPRC

input : Graph $G = (V \cup \{s, t\}, A)$, maximum iteration to execute random coloring algorithm $maxIter$, number of the solutions triggered early stop $nSol$

output: A set of routes with negative costs T

```
Initialization T = ∅ as solution set and k = 0
while k == maxIter or |T| > nSol do
    Generate a random coloring scheme φ_k : V → {1, ..., Q}
    Use Algorithm 3.3 to solve ESPPRC based on current color-coding φ_i
    Add routes with negative cost to T
    k = k + 1
end
return T
```

Irrespective of the number of repetitions $maxIter$, the random coloring algorithm has a non-zero probability of failure (i.e., it does not find any negative cost route even if one exists). To address this issue, we can either implement the de-randomized algorithm (which has the same asymptotic time complexity) as described in Section 4 of Alon et al. (1995) or any other exact algorithm (e.g., the pulse algorithm), as a "safe vault", to ensure that no more negative cost routes can be found in such cases. In our computational experiments,

we used the pulse algorithm as the safe vault as it was already implemented.

It is worth highlighting that the random coloring idea could be extended to other label-correcting algorithms for ESPPRC, as the label requires maintaining a binary vector recording the nodes of corresponding partial path visited. By randomly assigning nodes with a fixed set of colors, we can reduce the length of such vector and decrease the total number of labels to explore in the algorithm. One can also apply bidirectional search techniques to further improve the random coloring algorithm.

3.3.2.1 Non-robust Cuts

Valid inequalities can strengthen LP relaxations of integer programs and help to obtain integer solutions at the extreme points of LP relaxations. Poggi de Aragao and Uchoa (2003) propose to classify valid inequalities into "robust cuts" and "non-robust cuts". They apply "robust cuts" on the flow-based formulation of VRP (which can be transformed into RMP) and the cuts do not affect the complexity of the pricing problem as their associated dual variables only change the arc costs used in the pricing problem. Lysgaard et al. (2004) discuss various robust cuts for CVRP. On the other hand, non-robust cuts are applied directly on the LP relaxation of RMP and thus increase the complexity of the pricing problem as their associated dual variables cannot be incorporated into the arc costs. In this section, we will discuss how to incorporate non-robust cuts for our proposed algorithm.

Jepsen et al. (2008) introduced a family of valid inequalities named subset-row cuts for VRPTW and discussed how to handle the modified pricing problem via the label correcting algorithm for ESPPRC. The effective use of subset-row cuts yields better root-node bounds in the BCP approach and shortens the overall solution time. Therefore, subset-row cuts have been widely used column-generation-based approaches for VRPs (see, e.g., Baldacci et al., 2010, 2011; Pecin et al., 2017a).

The subset-row cuts are defined over route variables and are applied directly on the LP relaxation of RMP. Recall that a_{ip} is a binary coefficient indicating whether a route $p \in \tilde{P}$

visits a node $i \in V$. For any set $S \subset V$ and a multiplier $0 < k < 1$, a subset-row cut is given by

$$\sum_{p \in \tilde{P}} \left\lfloor k \sum_{i \in S} a_{ip} \right\rfloor x_p \leq \lfloor k|S| \rfloor . \tag{3.14}$$

Inequalities (3.14) are valid as they can be obtained by a Chvátal-Gomory rounding of constraints (3.5). Various combinations of $|S|$ and p yield effective subset-row cuts to improve the lower bounds given by the LP relaxation of RMP. For example, when $|S| = 3$ and $k = \frac{1}{2}$, cuts (3.14) are 3-subset-row cuts and when $|S| = 4$ and $k = \frac{2}{3}$, cuts (3.14) are 4-subset-row cuts. Subset-row cuts change the pricing problems. Let $\sigma_S \leq 0$ be the dual solutions associated with inequalities (3.14) when solving the LP relaxation of RMP. Then the reduced cost of a new column is given by $\bar{c}_p = \sum_{(i,j) \in p} (c_{ij} - \pi_j) - \sigma_S \left\lfloor k \sum_{i \in S} a_{ip} \right\rfloor$.

To incorporate the subset-row cuts into the random coloring algorithm, we follow the idea in Jepsen et al. (2008). In each iteration of column generation, we maintain a vector corresponding to the subset-row cuts with non-zero dual variables in the current solution to the LP relaxation of RMP, denoted as $\mathcal{S}$. This vector maintains counters for each subset-row cut: when a label extends to a node in a subset-row cut S, we add k to the value corresponding to that subset-row cut in the vector $\mathcal{S}$. When the value of any subset-row cut S (in vector $\mathcal{S}$) exceeds one, we update the cost by subtracting σ_S and subtract 1 from the coordinate S in vector $\mathcal{S}$.

The modification of the algorithm also changes the dominance rule. Let P'_{si} and P^*_{si} be two distinct paths from s to i with associated labels (R'_i, C'_i) and (R^*_i, C^*_i). Recall for the pricing problem without non-robust cuts, we say that P'_{si} dominates P^*_{si} if and only if $C'_i \leq C^*_i$, $n'_i \leq n^*_i$, $V'^j_i \leq V^{*j}_i$ for all $j \in \{1, 2, \ldots, Q\}$. With this modification, P'_{si} and P^*_{si} are associated with labels $(R'_i, \mathcal{S}'_i, C'_i)$ and $(R^*_i, \mathcal{S}^*_i, C^*_i)$, respectively and we state that P'_{si} dominates P^*_{si} if and only if $C'_i \leq C^*_i + \sum_{1 \leq s \leq |\mathcal{S}|: \mathcal{S}'_i[s] > \mathcal{S}^*_i[s]} \sigma_S$, $n'_i \leq n^*_i$, $V'^j_i \leq V^{*j}_i$ for all $j \in \{1, 2, \ldots, Q\}$ (Proposition 6 in Jepsen et al., 2008).

To keep the pricing problem tractable, only a small number of subset-row cuts are

included in the pricing problem. Pecin et al. (2017b) introduce a weak version of subsets row cuts called limited-memory subset-row cuts where each subset-row cut has a memory set, and the state counter of subset-row cut resets when a label extends to a node outside such a memory set. Our proposed algorithm can also be easily modified to incorporate limited-memory subset-row cuts following a similar idea.

3.3.2.2 Parallelization

The random coloring algorithm requires to explore different color-codings to increase the success probability of recovering all potential routes. In each iteration, a label correcting algorithm is executed based on the current color-coding, which is completely independent of all other iterations. For this reason, it is natural to perform a parallel implementation of the random coloring algorithm. We can invoke each iteration using parallel computer threads to accelerate the algorithm while maintaining the solution set T as global information. The number of threads, therefore, determines the number of color-coding iterations that can be implemented simultaneously.

3.4 Computational Experiments

We conduct numerical studies and demonstrate the performance of the proposed algorithms on tailored instances for VRPUD. We embed our proposed algorithms inside the column generation method for VRPUD. In experiments, we solve the root node linear relaxation of the set partitioning integer program (MP) and then use the generated columns to obtain integer solution of RMP. We conduct three sets of experiments: (i) a set of tailored instances from the Solomon's and Gehring & Homberger benchmark[1], (ii) selected unitary demand CVRP instances from CVRPLIB[2], and (iii) a multi-depot VRPUD which has potential applications in the patient-centered medical home.

[1] `https://www.sintef.no/projectweb/top/vrptw/`
[2] `http://vrp.galgos.inf.puc-rio.br/index.php/en/`

We implement our column generation method based on the conventional set partitioning formulation MP. We start the column generation with a series of heuristics that initialize the columns pool following a common practice (see, e.g., Feillet et al., 2004; Lozano et al., 2015). The heuristic is based on tabu search: we start with a set of feasible solutions (columns visiting only one node per vehicle) and then execute insertion and deletion operations until no further improvements can be made. After the initialization, we only solve subproblems as ESPPRC to generate columns.

After tuning in few preliminary tests, we choose our parameters in the algorithms as follows. For the pulse algorithm (Algorithm 3.2), we choose $\Delta = 1$, $\underline{Q} = 2$ and $nSol = 30$; For the random coloring algorithm (Algorithm 3.4), we choose $maxIter = 39$ and $nSol = 30$. For the random coloring algorithm, when the algorithm fails to find any routes with negative cost, we trigger a run of the pulse algorithm as a safe vault to ensure that no more routes of negative cost exist. As we discussed in the previous section, both algorithms can be implemented in parallel. In our tests, we implement the multi-thread versions of the proposed algorithms unless otherwise noted.

We code our algorithms in Java on a computer with two Intel Xeon E5-2630v4 processors with 20 cores each (40 total), and 128GB DDR4-2400 registered RAM. We used Gurobi 7.5.2 as the LP solver and mixed-integer linear programming solver.

3.4.1 Numerical Results on Single Depot VRPUD

3.4.1.1 Solomon and Gehring & Homberger Instances

The first set of test instances are modified from the Solomon benchmark with 100 customers[3] and Gehring & Homberger benchmark with up to 400 customers[4]. Both benchmark instances contain three types of node distributions: Type R instances where customers are randomly distributed, Type C instances where customers form several clusters, and Type

[3] https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/
[4] https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/

RC where some customers are randomly distributed while others are clustered. For each customer node in the test instances, we ignore its time windows and assign a unit demand. The travel distances between any two nodes are calculated as the Euclidean distance based on the coordinates given by the original data.

We solve the linear relaxation of MP using column generation on the test instances. We compare our proposed algorithms with the label correcting algorithm to ESPPRC from Feillet et al. (2004). As the original label correcting algorithm is implemented in serial, we compare it with **the serial implementation of proposed algorithms**. We test the performance of the algorithms on instances with number of customers ranging from 50 to 150. For instances with number of customers $|V| \leq 100$, we use the first $|V| + 1$ nodes from Solomon's instance and for $|V| > 100$, we use the first $|V| + 1$ nodes from Gehring & Homberger's instances: the first node represents the depot node in all benchmark instances. We consider $Q = 4$ for test instances. For the label correcting algorithm, the pulse algorithm, and the random coloring algorithm, we report the number of columns generated (nColumns), lower bound for MP (LB), and runtime in seconds for computing lower bound (Time). We set the time limit for column generation as 15 minutes for each test instance. Table 3.2 summarizes the computational results.

In Table 3.2, we observe the efficiency of the pulse algorithm on the test instances as its runtime to solve the linear relaxation of MP is significantly lower than the other two algorithms. Compared to the original label-correcting algorithm, the implementation with random coloring has significantly improved solution time. We observe that when the number of nodes increases, the label correcting algorithm encounters the curse of dimensionality as it fails to solve instances with more than 140 customer nodes. On the contrary, we can see the advantages of using the random coloring algorithm for the cases of larger instances as the problem can be consistently solved. The speed-up factor of the random coloring algorithm compared to the original label-correcting algorithm is between 2 and 10, and this factor increases for larger instances.

Table 3.2: Numerical results for proposed algorithms in serial implementation (Q=4)

Type	nNodes	Label Correcting			Pulse			Random Coloring		
		nColumns	LB	Time (s)	nColumns	LB	Time (s)	nColumns	LB	Time (s)
C	51	685	694.31	4.96	863	694.31	1.44	764	694.31	5.67
	61	764	902.46	20.13	1077	902.46	1.64	842	902.46	3.93
	71	920	1088.31	34.98	1171	1088.31	1.94	957	1088.31	6.13
	81	1027	1266.57	47.04	1394	1266.57	2.71	1047	1266.57	8.58
	91	1128	1437.82	37.77	1632	1437.82	3.73	1307	1437.82	13.42
	101	1193	1643.44	49.37	1710	1643.44	4.42	1376	1643.44	18.93
	111	1449	3211.32	367.57	2233	3211.32	8.06	1517	3211.32	61.02
	121	1868	3501.8	420.60	2673	3501.80	8.76	1777	3501.80	70.50
	131	2049	3798.77	557.21	2825	3798.77	10.30	1964	3798.77	106.54
	141	2053	4136.82	–	3089	4096.78	15.95	2063	4096.78	119.24
	151	2263	4442.25	–	3191	4398.63	13.61	2188	4398.63	137.89
R	51	569	916.81	4.08	707	916.81	1.42	754	916.81	3.67
	61	749	1029.79	14.00	843	1029.79	1.51	898	1029.79	7.10
	71	940	1235.64	28.42	987	1235.64	1.89	984	1235.64	10.50
	81	1038	1375.34	45.58	1234	1375.34	2.98	1132	1375.34	15.99
	91	1103	1510.59	52.83	1356	1510.59	3.51	1291	1510.59	26.61
	101	1267	1612.58	124.91	1456	1612.58	4.66	1481	1612.58	34.60
	111	1533	3508	283.43	1771	3508.00	5.26	1661	3508.00	61.99
	121	1829	3775.6	725.47	1847	3775.60	5.15	1834	3775.60	75.67
	131	1943	4123.27	–	2171	4107.18	6.58	1981	4107.18	93.00
	141	1946	4393.12	–	2538	4364.25	8.87	2082	4364.25	112.32
	151	2104	4669.05	–	2563	4624.60	10.40	2322	4624.60	150.79
RC	51	596	1124.15	4.01	596	1124.15	1.13	688	1124.15	2.85
	61	758	1349.72	8.78	836	1349.72	1.46	872	1349.72	7.07
	71	819	1488.27	25.76	1245	1488.27	2.28	920	1488.27	8.10
	81	991	1717.44	52.22	1247	1717.44	2.69	1130	1717.44	14.74
	91	1033	1871.52	71.30	1524	1871.53	4.09	1176	1871.53	21.53
	101	1244	1994.13	107.79	1684	1994.13	4.70	1394	1994.13	30.36
	111	1582	3459.84	117.58	2010	3459.84	7.45	1726	3459.84	54.83
	121	1688	3832.77	366.14	2224	3832.77	8.25	1892	3832.77	79.02
	131	1893	4190.07	–	2428	4174.07	9.74	2043	4174.07	87.58
	141	2039	4428.03	–	2646	4396.25	10.85	2145	4396.25	95.44
	151	2104	4731.66	–	2763	4685.09	13.53	2162	4685.09	121.90

– : runtime exceeds time limit of 15 minutes

As both pulse and the random coloring algorithms can be implemented in parallel, we also test both algorithms in their **parallel implementation**. Besides, we also investigate the optimality gap between the linear relaxation to the MP and the integral solution to the MP using the generated columns as well as their solution time. Both algorithms have been tested on instances with up to 600 customers for $Q = 3$ and 350 customers for $Q = 4$. For instances with a number of customers $|V| \leq 100$, we use the first $|V| + 1$ nodes from Solomon's instance and for $|V| > 100$, we use the first $|V| + 1$ nodes from Gehring & Homberger's instances. For both algorithms, we report the number of iterations to solve the column generation (nIter), number of columns generated in the column gen-

eration (nColumns), lower bound from the linear relaxation of MP (LB), upper bound of MP from the integral solution using the columns generated (UB), the optimality gap (Gap) computed as $\frac{UB-LB}{LB} \times 100\%$, and the runtime in seconds to solve for lower bound and upper bound (t_{LB} and t_{UB}, respectively). When solving MP as an integer program, we set the time limit as 2 hours.

Table 3.3: Numerical results for proposed algorithms for Type C instance in parallel implementation for Q=3

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	26	410	890.15	930.40	4.52%	0.46	0.96	20	365	890.15	932.80	4.79%	5.47	0.39
61	32	569	1157.03	1200.00	3.71%	0.28	2.12	19	422	1157.03	1203.90	4.05%	1.52	0.28
71	32	569	1369.50	1427.60	4.24%	0.65	1.69	21	483	1369.50	1429.50	4.38%	**0.47**	0.64
81	38	741	1624.08	1698.30	4.57%	1.00	1.72	26	565	1624.08	1697.70	4.53%	**0.65**	0.86
91	44	844	1839.15	1889.10	2.72%	0.84	2.50	27	620	1839.15	1894.40	3.00%	1.02	0.50
101	48	926	2104.92	2168.50	3.02%	0.96	5.31	29	739	2104.92	2188.10	3.95%	1.29	0.85
111	55	1253	4159.28	4333.50	4.19%	2.03	3.95	33	805	4159.28	4364.70	4.94%	10.81	6.85
121	70	1534	4540.81	4691.90	3.33%	5.43	19.88	42	948	4540.81	4710.60	3.74%	**2.77**	4.57
131	73	1620	4940.82	5075.80	2.73%	6.30	32.04	39	956	4940.82	5127.00	3.77%	**2.85**	0.94
141	77	1759	5336.18	5559.70	4.19%	85.41	16.84	45	1024	5336.18	5574.00	4.46%	**2.51**	5.04
151	75	1703	5735.61	5868.80	2.32%	8.56	1046.38	51	1177	5735.61	5897.40	2.82%	**3.07**	1.49
201	110	2534	14449.37	14973.70	3.63%	18.51	29.37	64	1508	14449.37	15004.00	3.84%	33.48	7.73
251	137	3254	17804.37	18502.20	3.92%	551.56	78.44	81	1938	17804.37	18496.50	3.89%	**17.97**	21.35
301	160	3993	21503.83	22139.50	2.96%	937.15	88.32	101	2407	21503.83	22231.00	3.38%	**22.38**	89.89
351	200	5099	24712.22	25418.30	2.86%	200.49	1361.62	121	2749	24712.22	25521.70	3.28%	**37.95**	64.29
401	233	6165	28365.05	29071.2	2.49%	118.71	93.49	129	3162	28365.05	29202.60	2.95%	**64.92**	214.02
451	267	7214	32325.08	33289.00	2.98%	163.18	–	152	3691	32325.08	33286.40	2.97%	**67.62**	483.10
501	313	8538	36256.18	36972.2	1.97%	236.90	–	170	4112	36256.18	36949.00	1.91%	**93.18**	103.67
551	357	9995	40037.55	40788.1	1.87%	325.94	–	185	4453	40037.55	40791.20	1.88%	**113.07**	79.38
601	403	11240	43394.23	44325.2	2.15%	441.61	–	213	4924	43394.23	44218.10	1.90%	**151.88**	1113.53

Tables 3.3–3.8 summarize the numerical results for the proposed algorithms on Type C, Type R, and Type RC instances with $Q = 3$ and $Q = 4$, respectively. We observe significant improvements in efficiency for both algorithms when they are implemented in parallel. We notice that when running in parallel, the random coloring algorithm outperforms the pulse algorithm (in lower bound runtime) in most of the instances with $Q = 3$ while their performances are mixed for instances with $Q = 4$: we highlight (in bold) the instances where the random coloring algorithm is faster than pulse in solving the LP relaxation. Notice that, although we allow both algorithms to stop early when the number of generated columns reach a preset limit, we still have one algorithm generating more columns than the other for some instances. It is because when the global number of generated columns reaches the preset bound, there are still some threads keeping a small set

Table 3.4: Numerical results for proposed algorithms for Type R instance in parallel implementation (Q=3)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	32	449	1123.86	1132.10	0.73%	0.82	0.41	19	416	1123.86	1132.00	0.72%	**0.46**	0.19
61	34	543	1267.60	1273.10	0.43%	0.90	0.26	21	490	1267.60	1279.20	0.92%	**0.55**	0.22
71	40	668	1528.22	1547.60	1.27%	1.02	0.60	25	563	1528.22	1544.20	1.05%	**0.80**	0.60
81	41	713	1702.30	1712.90	0.62%	1.27	0.49	28	612	1702.30	1713.90	0.68%	**0.75**	0.38
91	44	765	1870.56	1882.30	0.63%	1.55	0.54	32	719	1870.56	1881.20	0.57%	**0.97**	0.58
101	62	995	2003.33	2026.10	1.14%	2.56	2.29	35	751	2003.33	2022.60	0.96%	**1.30**	0.96
111	64	1135	4396.38	4438.00	0.95%	3.39	1.46	36	851	4396.38	4432.90	0.83%	**1.36**	0.66
121	65	1241	4741.20	4779.20	0.80%	3.45	1.25	41	938	4741.20	4794.10	1.12%	**1.80**	0.57
131	74	1385	5168.43	5247.40	1.53%	4.36	2.45	48	1073	5168.43	5237.10	1.33%	**2.34**	1.07
141	79	1478	5516.96	5595.50	1.42%	5.30	2.55	51	1136	5516.96	5599.30	1.49%	**2.79**	1.96
151	81	1655	5864.11	5942.40	1.34%	6.02	3.17	49	1171	5864.11	5936.00	1.23%	**3.31**	1.39
201	99	2200	15933.29	16134.90	1.27%	12.18	3.95	69	1671	15933.29	16180.60	1.55%	**7.50**	3.36
251	139	3180	19557.31	19697.40	0.72%	26.87	5.60	89	2130	19557.31	19735.40	0.91%	**14.41**	13.07
301	157	3772	23266.43	23527.70	1.12%	44.47	30.57	106	2575	23266.43	23493.00	0.97%	**23.71**	4.74
351	189	4606	27208.63	27405.00	0.72%	74.54	21.66	122	2948	27208.63	27492.90	1.04%	**36.04**	25.81
401	212	5346	31076.81	31297.2	0.71%	104.54	30.28	129	3152	31076.81	31464.20	1.25%	**49.13**	33.43
451	247	6374	34733.21	34940.5	0.60%	152.13	31.84	151	3564	34733.21	35034.60	0.87%	**68.29**	70.71
501	271	7107	38702.21	38935.1	0.60%	204.12	49.79	171	4058	38702.21	39003.00	0.78%	**92.18**	62.61
551	304	8107	42820.78	43048.4	0.53%	274.55	88.17	189	4637	42820.78	43112.20	0.68%	**121.94**	118.17
601	329	8866	46942.01	47237.8	0.63%	357.37	116.95	208	4933	46942.01	47271.60	0.70%	**153.99**	49.25

Table 3.5: Numerical results for proposed algorithms for Type RC instance in parallel implementation (Q=3)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	24	330	1457.28	1592.30	9.26%	0.59	0.28	17	386	1457.28	1569.70	7.71%	**0.27**	0.17
61	35	506	1725.75	1780.80	3.19%	0.79	0.89	21	467	1725.75	1790.80	3.77%	**0.50**	0.55
71	42	680	1886.43	1944.20	3.06%	1.07	1.51	23	530	1886.43	1949.90	3.36%	**0.69**	0.51
81	45	734	2161.80	2217.00	2.55%	1.31	1.18	25	588	2161.80	2223.00	2.83%	**0.84**	0.64
91	46	809	2363.60	2428.30	2.74%	1.53	0.92	28	630	2363.60	2409.50	1.94%	**1.27**	0.35
101	55	983	2517.67	2588.40	2.81%	2.09	1.62	32	756	2517.67	2576.80	2.35%	**1.44**	1.02
111	55	1122	4392.78	4453.10	1.37%	2.96	1.66	39	898	4392.78	4457.10	1.46%	**1.59**	1.49
121	70	1269	4850.89	4918.80	1.40%	3.58	2.46	43	956	4850.89	4930.90	1.65%	**1.82**	1.43
131	70	1476	5306.25	5347.70	0.78%	4.06	2.21	47	1040	5306.25	5373.30	1.26%	**2.24**	1.57
141	78	1579	5592.85	5646.20	0.95%	5.22	2.19	51	1179	5592.85	5641.00	0.86%	**2.85**	1.11
151	76	1639	5979.57	6055.80	1.27%	5.58	3.47	54	1253	5979.57	6054.20	1.25%	**3.25**	2.18
201	111	2378	15783.03	15951.60	1.07%	14.86	5.60	67	1625	15783.03	15968.20	1.17%	**6.97**	4.10
251	138	3067	19149.32	19518.70	1.93%	27.46	21.65	82	1989	19149.32	19470.60	1.68%	**12.95**	15.05
301	159	3719	22880.14	23109.60	1.00%	45.34	36.30	98	2287	22880.14	23202.90	1.41%	**21.98**	28.36
351	188	4504	26674.19	26990.40	1.19%	74.64	28.74	124	2877	26674.19	27043.60	1.38%	**36.52**	20.70
401	213	5276	30398.66	30724.5	1.07%	110.72	91.94	136	3253	30398.66	30826.10	1.41%	**50.13**	290.30
451	242	6125	33811.3	34190	1.12%	150.28	62.58	156	3713	33811.30	34247.50	1.29%	**70.96**	97.31
501	284	7423	37549.84	37858	0.82%	216.88	550.54	170	4068	37549.84	37931.20	1.02%	**90.72**	68.28
551	322	8268	41366.01	41757.2	0.95%	298.16	766.96	188	4486	41366.01	41806.10	1.06%	**120.01**	439.29
601	370	9913	44943.26	45307.6	0.81%	412.00	256.78	205	4895	44943.26	45409.90	1.04%	**150.73**	175.65

of paths pending to update to global column set. We decide to not waste those generated columns. In any case, the number of generated columns is much smaller than the total possible: for example, the maximum number of generated columns in instances with 301 nodes and $Q = 4$ was less than 7000 (whereas the total number possible is more than

Table 3.6: Numerical results for proposed algorithms for Type C instance in parallel implementation (Q=4)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	34	689	694.31	734.80	5.83%	0.92	0.96	19	1100	694.31	737.30	6.19%	4.43	1.61
61	45	974	902.46	954.00	5.71%	1.06	2.12	21	1321	902.46	954.80	5.80%	1.06	2.83
71	54	1096	1088.31	1139.60	4.71%	1.42	1.69	23	1398	1088.31	1153.40	5.98%	**1.18**	3.56
81	57	1312	1266.57	1310.00	3.43%	1.77	1.72	28	1732	1266.57	1318.40	4.09%	2.55	2.32
91	71	1646	1437.82	1523.10	5.93%	2.81	2.50	31	1925	1437.82	1535.30	6.78%	**2.51**	4.37
101	78	1769	1643.44	1753.10	6.67%	3.42	5.31	32	2057	1643.44	1759.50	7.06%	**3.02**	5.26
111	89	2232	3211.32	3391.90	5.62%	5.54	3.95	35	2346	3211.32	3384.50	5.39%	6.80	4.10
121	104	2537	3501.80	3742.70	6.88%	6.58	19.88	38	2552	3501.80	3781.40	7.98%	**6.29**	16.22
131	115	2772	3798.78	4061.90	6.93%	7.91	32.04	40	2693	3798.77	4047.80	6.56%	**7.79**	13.21
141	119	2992	4096.78	4313.20	5.28%	9.52	16.84	43	3037	4096.78	4385.10	7.04%	9.62	21.70
151	120	3111	4398.63	4674.10	6.26%	10.54	1046.38	47	3192	4398.63	4675.80	6.30%	12.71	29.04
201	168	4119	11201.63	11809.70	5.43%	25.45	29.37	60	4066	11201.63	11717.30	4.60%	44.11	24.99
251	228	5854	13706.13	14276.00	4.16%	52.73	78.44	79	5216	13706.12	14446.70	5.40%	**47.64**	300.66
301	263	6839	16431.13	16970.10	3.28%	86.15	88.32	94	6115	16431.13	17048.40	3.76%	**76.08**	44.35
351	313	8570	18858.48	19663.30	4.27%	137.27	1361.62	106	6919	18858.48	19600.90	3.94%	**110.04**	79.37

Table 3.7: Numerical results for proposed algorithms for Type R instance in parallel implementation (Q=4)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	36	582	916.81	937.50	2.26%	0.92	0.64	20	1277	916.81	933.40	1.81%	**0.72**	0.80
61	47	820	1029.79	1039.40	0.93%	1.12	0.44	22	1446	1029.79	1051.50	2.11%	**1.06**	1.18
71	55	1048	1235.64	1253.90	1.48%	1.55	1.17	26	1643	1235.64	1246.20	0.85%	1.61	1.48
81	59	1123	1375.34	1382.60	0.53%	1.92	0.79	29	1906	1375.34	1389.60	1.04%	2.25	1.97
91	64	1239	1510.59	1528.80	1.21%	2.50	1.53	34	2163	1510.59	1541.20	2.03%	3.29	3.58
101	75	1426	1612.58	1630.90	1.14%	3.41	1.74	35	2261	1612.58	1633.00	1.27%	4.25	2.94
111	86	1736	3508.00	3563.30	1.58%	5.12	18.96	38	2426	3508.00	3555.90	1.37%	5.45	2.70
121	96	2088	3775.60	3850.20	1.98%	5.91	2.45	40	2665	3775.60	3847.10	1.89%	7.09	4.65
131	106	2306	4107.18	4199.40	2.25%	7.56	4.12	43	2864	4107.18	4232.00	3.04%	8.75	16.92
141	114	2613	4364.25	4438.80	1.71%	9.11	4.81	47	3134	4364.25	4473.90	2.51%	10.83	15.05
151	115	2657	4624.60	4702.80	1.69%	9.86	4.72	53	3431	4624.60	4701.90	1.67%	13.79	6.09
201	153	3661	12484.12	12743.90	2.08%	24.31	33.57	66	4186	12484.12	12719.80	1.89%	28.68	11.54
251	195	4710	15257.69	15458.90	1.32%	46.69	44.11	80	5364	15257.69	15548.40	1.91%	49.92	49.07
301	252	6412	18099.94	18393.50	1.62%	85.89	58.86	97	6221	18099.94	18383.30	1.57%	**81.73**	70.70
351	279	7341	21093.32	21449.20	1.69%	128.08	160.38	115	7498	21093.32	21413.80	1.52%	**124.27**	255.97

7.9 billion). Furthermore, despite the column generation yielding the same lower bound (the optimal LP value) by using two different algorithms for solving the pricing problem, they return different upper bounds when solving the RMP as an integer program using the generated columns. Comparing among different types of instances with the same number of nodes in the underlying network, the optimality gap for column generation method in Type C instance (6% on average) is greater than the ones in the other two types (less than 3% on average). Overall, the optimality gap between the integral solution to RMP and the solution to the linear relaxation of MP is small and shows a decreasing trend when the num-

Table 3.8: Numerical results for proposed algorithms for Type RC instance in parallel implementation (Q=4)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	31	568	1124.15	1257.10	11.83%	0.81	0.53	18	989	1124.15	1259.50	12.04%	**0.59**	1.84
61	43	799	1349.72	1429.60	5.92%	0.96	2.25	22	1328	1349.72	1432.00	6.10%	1.05	2.82
71	58	1096	1488.27	1554.50	4.45%	1.55	1.56	24	1512	1488.27	1556.80	4.60%	**1.47**	2.51
81	60	1097	1717.44	1772.70	3.22%	1.99	2.40	27	1783	1717.44	1779.60	3.62%	2.00	3.28
91	71	1374	1871.52	1946.20	3.99%	2.78	2.55	32	1966	1871.53	1964.10	4.95%	3.02	5.21
101	79	1589	1994.13	2060.70	3.34%	4.12	2.58	32	1986	1994.13	2087.20	4.67%	**3.75**	4.70
111	90	1905	3459.84	3558.10	2.84%	4.91	7.63	38	2432	3459.84	3534.50	2.16%	5.24	4.38
121	97	2182	3832.77	3895.90	1.65%	5.75	2.52	40	2628	3832.77	3924.50	2.39%	6.44	5.98
131	105	2364	4174.07	4304.10	3.12%	7.24	19.31	43	2683	4174.07	4273.80	2.39%	8.10	9.04
141	110	2475	4396.25	4446.50	1.14%	8.63	3.34	49	3093	4396.25	4513.60	2.67%	10.91	40.73
151	114	2740	4685.09	4776.30	1.95%	9.96	39.34	50	3245	4685.09	4783.70	2.10%	12.38	17.39
201	164	3916	12374.48	12629.00	2.06%	26.26	34.20	64	4216	12374.48	12623.00	2.01%	26.55	31.33
251	207	5124	14890.03	15201.20	2.09%	49.69	134.37	82	5376	14890.02	15129.00	1.60%	49.78	52.16
301	235	6121	17703.31	17953.00	1.41%	81.26	76.01	92	5957	17703.31	18060.50	2.02%	**78.04**	442.43
351	304	7985	20648.60	20950.60	1.46%	142.38	429.40	109	7138	20648.60	21035.10	1.87%	**113.24**	836.24

ber of nodes in the network increases. Therefore, we conclude that the column generation approach could provide solutions with good quality for the large instances.

We also study the effect of the parallelization. When running in parallel, the speedup for the pulse algorithm is limited while the random coloring algorithm gets significantly boosted because the runs for different color-codings are completely independent. Figure 3.2 shows the effect of the parallelization for the two algorithms by plotting the average speedup factor across three different types of instances with $Q = 4$ on 50–150 nodes. As shown in the figure, we can observe a significant improvement from using the random coloring algorithm as the speedup factor ranges from 5 to 14 using two 20-core processors. On the other hand, the speedup factor for pulse algorithm is limited.

We also conduct numerical experiments with different vehicle capacities. The results for the proposed algorithms on the instances with $Q = 5$ and $Q = 6$ are attached in the appendix. It shows that both algorithms perform similarly to the case of $Q = 3$ and $Q = 4$, but we observe that the pulse algorithm becomes more efficient than the random coloring algorithm as Q increases.

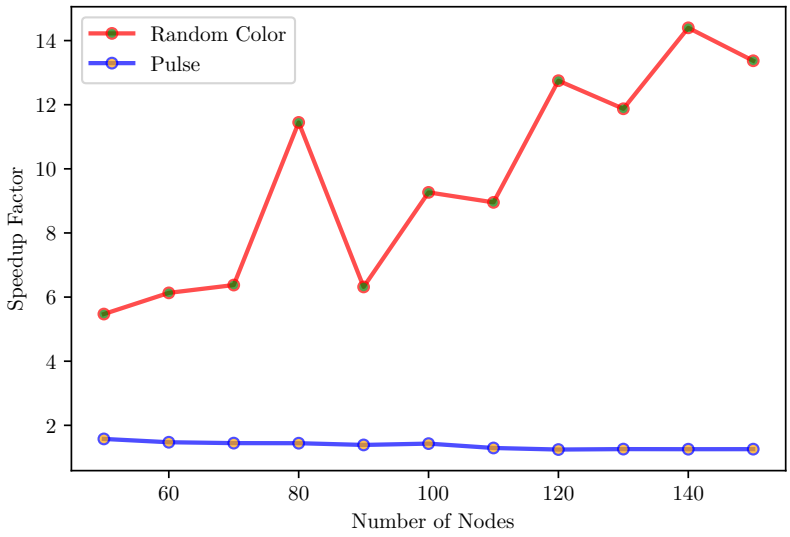


Figure 3.2: Average speedup factor in parallel implementation for instances with $Q = 4$

3.4.1.2 Unitary Demand CVRP X-instances

As a special case of CVRP, there are some unitary demand CVRP instances that have been proposed in the literature. In particular, Uchoa et al. (2017) propose a set of new benchmark instances for the CVRP. The new set of instances are generated on a $[0, 1000] \times [0, 1000]$ two-dimensional space with different settings on the number of customers, the capacity of the vehicle, depot location, and demand distribution. Out of 100 instances, 16 are generated as unitary demand instances. The original capacity of the vehicle ranges from 3 to 23 in those instances. To demonstrate our proposed algorithms, we test both algorithms (pulse and random coloring) on those instances with modified capacity $Q = 3$ and $Q = 4$. We use the same node location of the original instances and compute the distance of two nodes as its Euclidean distance rounding to the nearest integer. In our experiments, we report the lower bound from the linear relaxation of MP (LB), the upper bound of MP (UB) obtained using all columns generated through the column generation, the optimality gap (Gap) computed as $\frac{UB-LB}{LB} \times 100\%$, and the runtime in minutes to solve the lower bound and upper bound (t_{LB}, t_{UB}), respectively. To avoid the excessive time needed to solve

the restricted integer program for MP, we only use the columns that were active (with a non-zero solution to the linear relaxation of RMP) during the final 50 iterations of column generation as input. The time limit for solving the integer program is set as 1 hour.

Table 3.9: Numerical results for unitary X instances with Q=3

Instance	n	Pulse					Random Coloring				
		LB	UB	Gap	t_{LB} (min)	t_{UB} (min)	LB	UB	Gap	t_{LB} (min)	t_{UB} (min)
X-n120-k6	120	60956.67	61748	1.28%	0.09	0.01	60956.67	62287	2.14%	0.25	0.01
X-n157-k13	157	56728.5	57281	0.96%	0.17	0.02	56728.5	57482	1.31%	**0.08**	0.01
X-n181-k23	181	59784.83	60329	0.90%	0.22	0.09	59784.83	60202	0.69%	**0.11**	0.02
X-n219-k73	219	117208	118539	1.12%	0.46	0.18	117208	118413	1.02%	**0.17**	0.02
X-n237-k14	237	124225.3	125293	0.85%	0.58	0.16	124225.3	125381	0.92%	**0.21**	0.21
X-n275-k28	275	56686.28	57525	1.46%	0.69	0.51	56686.28	57466	1.36%	**0.29**	0.04
X-n317-k53	317	150725.5	154535	2.47%	1.84	0.01	150725.5	153712	1.94%	**0.53**	0.04
X-n331-k15	331	180765.8	186265	2.95%	1.58	0.02	180765.7	184330	1.93%	**0.58**	0.03
X-n376-k94	376	193335.5	200333	3.49%	2.47	1.86	193335.5	197163	1.94%	**0.81**	0.03
X-n439-k37	439	116065.3	120058	3.33%	2.76	0.20	116065.3	120423	3.62%	**1.12**	0.41
X-n502-k39	502	277983	295325	5.87%	9.93	0.31	277983	287670	3.37%	**2.01**	2.53
X-n548-k50	548	287791.2	301644	4.59%	8.28	0.03	287791.2	301186	4.45%	**2.38**	0.04
X-n655-k131	655	172978.4	181083	4.48%	11.77	0.16	172978.4	181116	4.49%	**3.86**	0.13
X-n801-k40	801	415751.4	436178	4.68%	31.68	–	415751.4	443677	6.29%	**6.46**	0.02
X-n856-k95	856	240465.8	253476	5.13%	29.26	0.05	240465.8	259893	7.48%	**7.03**	0.02
X-n957-k87	957	275862	290521	5.05%	49.88	0.44	275861.9	297325	7.22%	**9.70**	0.03

–: solution time reaches 60-minute time limit.

Table 3.10: Numerical results for unitary X instances with Q=4

Instance	n	Pulse					Random Coloring				
		LB	UB	Gap	t_{LB} (min)	t_{UB} (min)	LB	UB	Gap	t_{LB} (min)	t_{UB} (min)
X-n120-k6	120	47058.5	48428	2.83%	0.17	0.02	47058.5	47579	1.09%	**0.16**	0.04
X-n157-k13	157	43463.17	45774	5.05%	0.45	0.70	43463.17	44644	2.64%	**0.32**	0.14
X-n181-k23	181	45936.48	46853	1.96%	0.42	0.43	45936.47	46391	0.98%	0.47	0.25
X-n219-k73	219	89872.02	92230	2.56%	0.94	0.31	89872.02	91903	2.21%	**0.72**	1.86
X-n237-k14	237	95112.91	96156	1.08%	1.17	0.03	95112.91	95955	0.88%	**0.91**	0.39
X-n275-k28	275	43920.78	45535	3.55%	1.42	0.37	43920.78	45043	2.49%	**1.13**	2.03
X-n317-k53	317	114502	119402	4.10%	4.31	0.01	114502	117282	2.37%	**1.94**	6.51
X-n331-k15	331	137773.5	144045	4.35%	3.38	0.70	137773.5	140146	1.69%	**2.08**	3.67
X-n376-k94	376	147298.8	155445	5.24%	4.91	1.14	147298.8	150636	2.22%	**2.98**	3.75
X-n439-k37	439	89463.81	94625	5.45%	4.73	0.15	89463.81	93708	4.53%	**3.96**	8.98
X-n502-k39	502	209866.2	226300	7.26%	22.23	1.95	209866.2	217109	3.34%	**7.62**	1.68
X-n548-k50	548	218752.4	230237	4.99%	16.87	0.51	218752.4	228594	4.31%	**8.38**	6.46
X-n655-k131	655	131550.2	138081	4.73%	27.13	0.99	131550.2	136467	3.60%	**14.86**	8.34
X-n801-k40	801	315079.9	336845	6.46%	92.76	1.02	315079.9	334016	5.67%	**43.76**	14.54
X-n856-k95	856	183654.7	196487	6.53%	45.18	0.17	183654.7	196289	6.44%	**28.17**	40.78
X-n957-k87	957	210461.8	227655	7.55%	96.71	0.83	210461.8	225333	6.60%	**58.50**	7.45

Tables 3.9 and 3.10 summarize the numerical results for instances derived from unitary demand CVRP X-instances in Uchoa et al. (2017). When the capacity of the vehicle is small, both algorithms are capable of solving the root node relaxation of instances with

up to 957 customer nodes within a reasonable amount of time. Comparing between two algorithms, the random coloring algorithm, in general, outperforms the pulse algorithm in the LP solution speed: it is approximately 3–5 times faster when $Q = 3$ and two times faster when $Q = 4$. In our preliminary test, when using all columns generated during column generation approach to solve the RMP, the solution time to find optimal integer solution would be very long (more than 2 hours) for instances with more than 300 customer nodes. The solution speed for the integer program was significantly improved by using only the columns generated during the final 50 iterations of column generation. The optimality gaps obtained by two algorithms are similar and are ranging 0.69%–7.55%. We notice that the optimality gap obtained at the root node increases as the size of instances increases.

We also test modified X-instances with $Q = 5$ and $Q = 6$. The results have been attached in the appendix. We observe that the pulse algorithm is more efficient when solving the problem with larger vehicle capacity.

3.4.2 Numerical Results on Multi-depot VRPUD

In this section, we discuss an application of VRPUD in a patient-centered medical home system where caregivers route one or multiple fleets of vehicles to serve/treat patients in their homes. Patient-centered medical home has been considered as an effective and economical way to serve patients and is experiencing a fast-growing development (Musich et al., 2015). In 2012, over 4.7 million patients received services from about 12,000 registered home health agencies (Harris-Kojetin et al., 2013) and nowadays patient-centered medical home makes up more than 35% of post-acute care in the market.

According to Fikar and Hirsch (2017), different objectives and constraints of the patient-centered medical home problem have been studied. They summarize that possible objectives are total traveling time, operational cost, total wait time, total overtime, workload balance, number of tasks, etc.; and possible constraints include time windows, skill requirements, working time regulations, breaks, uncertainties, and so on (see, e.g., Allaoua

et al., 2013; Bachouch et al., 2011; Dohn et al., 2009; Fernandez et al., 1974; Lanzarone and Matta, 2014). In this section, we model the patient-centered medical home problem as a VRPUD motivated by the observation that the average number of patients that can be visited by a crew is small during one working period. We generalize the problem as a multi-depot VRPUD allowing caregivers to operate the system with multiple bases to start and end their service routes.

Our proposed solution approach can be easily extended to the multi-depot VRPUD. Let D be the set of the depots, and $\tilde{P}^d$ be the set of some routes starting and ending at depot $d \in D$. Using the same parameters and decision variables defined in Section 3.2, the restricted master problem of column generation for the multi-depot VRPUD has the following formulation.

$$\textbf{(MD-RMP)} \text{ minimize: } \sum_{d \in D} \sum_{p \in \tilde{P}^d} c_p x_p \tag{3.15}$$

$$\text{subject to: } \sum_{d \in D} \sum_{p \in \tilde{P}^d} a_{ip} x_p = 1 \qquad \forall\, i \in V, \tag{3.16}$$

$$x_p \in \{0, 1\} \qquad \forall\, p \in \tilde{P}^d,\ d \in D, \tag{3.17}$$

After solving the linear relaxation of the **MD-RMP**, we need to solve the pricing problems using the dual solutions corresponding to each constraint (3.16). However, as we have multiple depots, we need to generate routes for the vehicles based at the different depots, i.e., solving multiple pricing problems based on different depots.

The test instances for multi-depot VRPUD are based on the most updated United States Census data for Wayne County in Michigan (see United States Census Bureau 2010[5]). The census data divides Wayne County into 610 different census tracts, and each contains the geographical information (longitude and latitude of the geographical center). The detailed

[5] https://www2.census.gov/

reference map can be found at *Michigan 2010 Census - Census Tract Reference Maps*[6]. We assume that its geographical center represents each census tract and construct a corresponding network with 610 nodes. In addition, we use the geographical information of the top five hospitals in Wayne County as the depot nodes. The five hospitals are (1) Harper University Hospital, (2) Henry Ford Hospital, (3) DMC Sinai-Grace Hospital, (4) Henry Ford Wyandotte Hospital, and (5) Beaumont Hospital-Wayne. The distribution of the hospitals is shown in Figure 3.3.

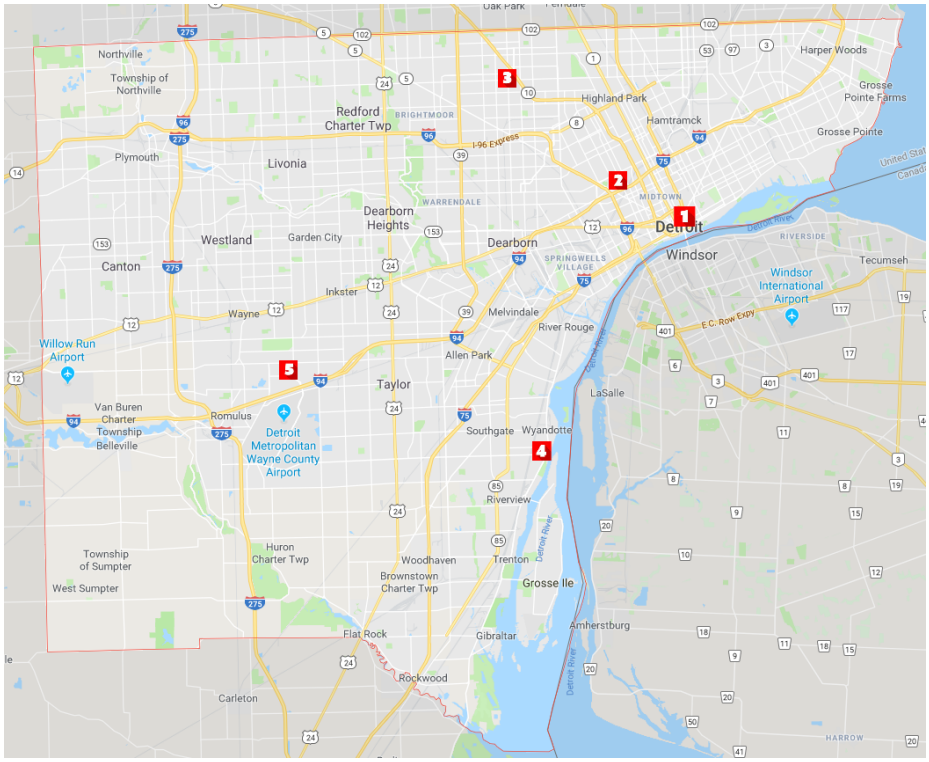


Figure 3.3: Distribution of hospitals in Wayne County

The travel time between any of two nodes is calculated through Haversine Equation[7]: for any two points with longitude φ_1, φ_2 and latitude λ_1, λ_2, the distance is given by:

$$d((\varphi_1, \lambda_1), (\varphi_2, \lambda_2)) = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

[6] https://www2.census.gov/geo/maps/dc10map/tract/st26_mi/c26163_wayne/
[7] https://en.wikipedia.org/wiki/Haversine_formula

In this experiment, we assume that vehicles start and end the route at the same depot (hospital) while covering all the patients. We test both proposed algorithms on the instances with the number of patient nodes ranging from 100 to 500. We test against the instances with 1, 3, or 5 depots and use the parallel implementation of the algorithms. We consider $Q = 4$ in our test instances. For any instance with $|V|$ patient nodes, we randomly pick $|V|$ data points from 610 census tracts as patient nodes. We report the number of iterations for solving the pricing problem (nIter), number of columns generated (nColumns), lower bound from the linear relaxation of **MD-RMP** (LB), upper bound of **MD-RMP** from the integral solution using the columns generated (UB), the optimality gap (Gap) computed as $\frac{UB-LB}{LB} \times 100\%$, and the runtime of computing lower bound and upper bound in seconds (t_{LB} and t_{UB}, respectively). When solving **MD-RMP** as an integer program, we set the time limit as 2 hours.

Table 3.11: Numerical result for the proposed algorithm on patient-centered medical home instances

nNode	nDepot	Pulse							Random Coloring						
		nIter	nColumns	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nColumns	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
100	1	99	2409	971.62	991.92	2.09%	6.50	5.45	31	1728	971.62	990.96	1.99%	14.71	2.75
	3	57	3085	759.23	778.90	2.59%	8.17	3.83	18	2859	759.23	778.34	2.52%	**6.56**	3.50
	5	25	2036	486.91	503.24	3.35%	5.88	1.94	13	2323	486.91	498.84	2.45%	7.33	2.11
150	1	148	3826	1398.61	1423.14	1.75%	16.15	17.76	45	2528	1398.61	1416.59	1.29%	18.30	5.77
	3	78	4893	1091.60	1101.16	0.88%	22.66	4.95	26	3796	1091.60	1104.57	1.19%	**20.90**	6.90
	5	44	3370	669.27	681.97	1.90%	20.52	4.81	14	3138	669.27	675.65	0.95%	**17.63**	2.42
200	1	211	5671	1852.60	1875.21	1.22%	40.26	178.24	63	3603	1852.60	1872.53	1.08%	**28.58**	72.69
	3	112	7365	1428.08	1441.64	0.95%	57.68	32.00	34	5325	1428.08	1442.83	1.03%	**44.47**	34.55
	5	46	4279	852.30	867.48	1.78%	37.73	6.22	21	3888	852.30	866.44	1.66%	41.66	6.51
250	1	303	8408	2274.12	2293.87	0.87%	89.82	63.99	76	4339	2274.12	2298.09	1.05%	**50.53**	134.54
	3	147	9746	1748.24	1764.20	0.91%	118.44	49.34	38	5838	1748.24	1767.79	1.12%	**71.99**	109.65
	5	65	5494	1033.53	1055.22	2.10%	84.31	68.46	22	5044	1033.53	1057.90	2.36%	**64.54**	38.96
300	1	388	11226	2736.67	2751.82	0.55%	164.92	51.40	93	5319	2736.67	2762.36	0.94%	**80.11**	269.75
	3	187	12002	2094.00	2104.35	0.49%	214.41	32.31	50	7077	2094.00	2103.98	0.48%	**123.96**	21.77
	5	79	6902	1235.72	1255.16	1.57%	147.11	42.35	27	5747	1235.72	1251.39	1.27%	**112.21**	28.78
350	1	477	13727	3173.38	3188.32	0.47%	274.14	530.63	112	6257	3173.38	3192.02	0.59%	**127.75**	83.16
	3	226	15152	2418.36	2429.37	0.46%	346.35	131.20	56	8242	2418.36	2435.92	0.73%	**182.28**	248.34
	5	93	8255	1404.23	1418.66	1.03%	233.33	38.64	33	6763	1404.23	1417.92	0.98%	**172.39**	51.80
400	1	596	17313	3580.96	3599.36	0.51%	431.60	411.34	125	7165	3580.96	3609.41	0.79%	**179.27**	3863.39
	3	260	17964	2719.65	2738.64	0.70%	505.62	213.47	66	9608	2719.65	2744.61	0.92%	**272.19**	1655.30
	5	103	9253	1586.06	1603.77	1.12%	327.58	91.37	37	7804	1586.06	1605.40	1.22%	**241.85**	66.93
450	1	682	19942	3981.56	3997.29	0.40%	611.65	2502.78	142	7745	3981.56	4011.20	0.74%	**253.12**	895.71
	3	298	20372	3011.01	3026.54	0.52%	713.83	727.64	79	10318	3011.01	3029.06	0.60%	**402.46**	386.21
	5	126	11209	1799.96	1817.65	0.98%	499.46	120.01	38	8520	1799.96	1817.04	0.95%	**310.38**	100.41
500	1	836	24471	4439.35	4455.34	0.36%	913.61	2437.67	156	8876	4439.35	4459.56	0.46%	**328.04**	271.89
	3	349	23711	3370.00	3384.41	0.43%	1022.49	169.83	80	11697	3370.00	3392.79	0.68%	**496.09**	1437.17
	5	129	12494	2006.88	2023.09	0.81%	622.25	304.67	42	9355	2006.88	2029.61	1.13%	**410.50**	940.39

Table 3.11 summarizes the numerical results of the proposed algorithms on the multi-

depot VRPUD embedded in the patient-centered medical home problem. As the multi-depot VRPUD requires us to solve the pricing problem based on each depot, the solution time increases when we have three depots instead of one. However, the increase factor is less than 3. Surprisingly, when the number of depots increases to five, the solution time for the instance is shorter than the cases where three depots allowed. We believe that the involvement of more depots, especially new depots (Hospital 4 and 5) separated away from the existing ones in our test instance, would reduce the empirical complexity of the problem. Between using pulse algorithm and random coloring algorithm to solve the column generation, the random coloring algorithm is more efficient in solving multi-depot instances especially when the number of patient is large. The optimality gap yielded by two algorithms are small (less than 2% in general). Throughout the experiments, our results show that both algorithms, within a reasonable amount of time, are capable of solving large multi-depot VRPUD instances containing up to 500 patient nodes, which is a practical amount under the context of a patient-centered medical home system in Wayne County. Furthermore, as shown in the numerical results, the optimality gap using the column generation approach is negligibly small considering the size of the instance.

We also study the solution routes computed from the column generation using two different pricing algorithms. For each instance with the different number of patient nodes (nNode) and hospitals (nDepot), we report the number of solution routes (nRoute) and their average cost (AvgCost) based at each depot. Table 3.12 and 3.13 summarize the solution results. Comparing the instances with the same number of customer nodes (patients) but a different number of depots (hospitals), we notice the total number of routes used to cover the patients are similar as most of the routes contain four patients. However, the average cost of each route reduces 30%-40% (from approximately 40 to approximately 25) as the number of depots increases from 1 to 3. Further reduction repeats, though diminishing, is observed as we increase the number of depots to 5, reducing the average cost per route from 25 to 20. An example solution for instances with 500 patients and five hospitals has

Table 3.12: Solution summary of multi-depot VRPUD with pulse pricing algorithm

nNode	nDepot	Depot 1		Depot 2		Depot 3		Depot 4		Depot 5	
		nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost
100	1	25	39.88	–	–	–	–	–	–	–	–
	3	6	24.69	7	26.80	13	35.17	–	–	–	–
	5	5	23.39	5	14.63	6	18.26	2	15.51	9	20.06
150	1	38	38.01	–	–	–	–	–	–	–	–
	3	8	23.70	10	22.28	20	35.20	–	–	–	–
	5	8	21.07	7	13.79	8	16.76	3	17.93	14	16.92
200	1	51	37.08	–	–	–	–	–	–	–	–
	3	12	26.20	14	21.56	25	34.15	–	–	–	–
	5	9	20.41	10	14.94	11	16.25	5	15.60	16	18.22
250	1	63	36.67	–	–	–	–	–	–	–	–
	3	15	22.29	17	21.22	32	33.94	–	–	–	–
	5	14	20.09	10	12.76	13	14.92	7	17.28	20	17.31
300	1	76	36.58	–	–	–	–	–	–	–	–
	3	19	24.30	18	19.60	39	33.35	–	–	–	–
	5	14	20.20	13	12.72	16	15.85	10	14.03	25	16.96
350	1	88	36.56	–	–	–	–	–	–	–	–
	3	23	24.13	19	19.66	46	33.10	–	–	–	–
	5	16	18.86	15	13.72	20	15.42	9	14.64	29	16.84
400	1	100	36.21	–	–	–	–	–	–	–	–
	3	23	22.10	25	21.56	53	32.29	–	–	–	–
	5	19	18.39	17	12.11	24	15.18	11	14.52	33	16.20
450	1	113	35.56	–	–	–	–	–	–	–	–
	3	27	22.31	26	20.54	60	31.67	–	–	–	–
	5	22	18.88	19	13.07	25	14.89	12	15.80	36	16.86
500	1	126	35.55	–	–	–	–	–	–	–	–
	3	32	22.05	27	18.99	67	32.74	–	–	–	–
	5	25	18.39	21	13.22	28	15.47	13	15.16	40	16.71

been displayed in Figure 3.4.

Table 3.13: Solution summary of multi-depot VRPUD with random coloring pricing algorithm

nNode	nDepot	Depot 1		Depot 2		Depot 3		Depot 4		Depot 5	
		nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost	nRoute	avgCost
100	1	25	39.3	–	–	–	–	–	–	–	–
	3	6	26.5	6	25.44	13	35.66	–	–	–	–
	5	5	23.18	4	15.93	6	18.07	2	14.45	9	20.06
150	1	38	37.07	–	–	–	–	–	–	–	–
	3	9	22.89	9	22.18	20	34.88	–	–	–	–
	5	7	20.7	8	16.82	8	17.59	3	15.22	12	17.64
200	1	50	37.31	–	–	–	–	–	–	–	–
	3	11	25.16	13	20.44	26	34.5	–	–	–	–
	5	10	20.62	9	13.54	11	16.79	5	15.51	16	17.11
250	1	63	36.3	–	–	–	–	–	–	–	–
	3	16	22.91	15	21.18	32	33.6	–	–	–	–
	5	13	19.62	11	13.34	13	15.46	6	15.32	20	17.81
300	1	75	36.62	–	–	–	–	–	–	–	–
	3	19	23.59	18	19.73	39	33.33	–	–	–	–
	5	14	19.5	13	12.81	17	16.81	9	15.26	23	16.64
350	1	88	36.16	–	–	–	–	–	–	–	–
	3	20	23.55	22	20.32	46	32.86	–	–	–	–
	5	16	18.64	15	13.45	19	15.21	9	14.64	29	16.95
400	1	100	35.94	–	–	–	–	–	–	–	–
	3	24	23.44	23	20.04	53	32.2	–	–	–	–
	5	18	19.18	17	13.02	22	14.65	10	15.22	33	16.81
450	1	113	35.33	–	–	–	–	–	–	–	–
	3	26	21.99	28	20.58	59	31.72	–	–	–	–
	5	21	18.72	20	12.93	25	15.16	11	15.48	36	16.95
500	1	125	35.6	–	–	–	–	–	–	–	–
	3	31	22.37	29	20.1	65	32.38	–	–	–	–
	5	24	18.95	21	12.93	27	14.47	12	15.24	42	17.07

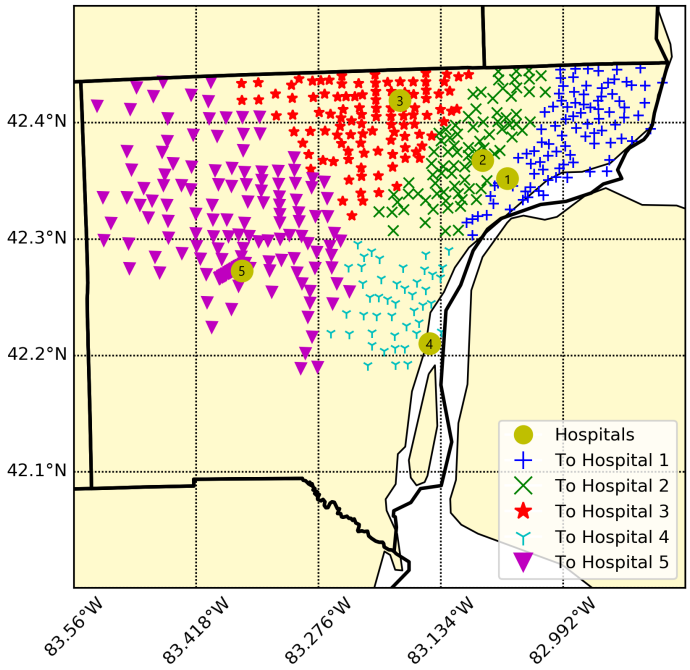


Figure 3.4: Example solution of assignments to the instance with 500 patients and 5 hospitals

3.5 Concluding Remarks

In this chapter, we studied VRPUD, a special case of CVRP where each customer has a unit demand, and applied the column generation method to solve the problem. To efficiently solve the exact pricing problem (ESPPRC) in the column generation approach, we proposed two parallel pricing algorithms: an extension of the pulse algorithm from Lozano and Medaglia (2013) with a new bounding scheme on the load of the vehicle and the random coloring algorithm based on the color-coding approach from Alon et al. (1995). Both algorithms could be implemented in parallel to achieve better efficiency.

We conducted numerical tests to evaluate these pricing approaches. In terms of runtime, the random coloring algorithm was typically faster for small capacities, but the pulse algorithm was faster as the capacity increased. Using the generated columns to compute the integer solution, we observed that both algorithms found high-quality integer solutions. We also investigated the multi-depot VRPUD inspired by the application of the patient-centered medical home delivery. The numerical study showed that both pricing algorithms could be applied to solve instances with up to 500 nodes and 5 depots within a reasonable time (1 hour). By allowing more depots, we observed a noticeable decrease in the overall traveling cost.

CHAPTER 4

Location Design and Relocation of a Mixed Car-Sharing Fleet with a CO_2 Emission Constraint

4.1 Introductory Remarks

Carsharing has become an increasingly popular means of transit over the last decade. As of 2014, an estimated 1.3 million members participated in North American carsharing programs (Shaheen and Cohen, 2014). Meanwhile, municipal governments, non-profit organizations, and for-profit companies alike have been endeavoring to create and expand carsharing programs. In 2015, the Seattle Department of Transportation reported that since launching a carsharing program in 2012, nearly 40,000 members joined the program and that 3% to 4% (1,200 to 1,600) members have given up their vehicles. A non-profit organization, City Carshare, launched in 2001, reported that as a direct result of its program, 17,000 vehicles were removed from Bay Area neighborhoods and that the drivers there drove 140 million fewer miles combined. For-profit companies, such as Zipcar and Car2Go, seek revenue as their first priority, but tout resourceful urban living as a part of their respective missions and keep careful tabs on the environmental gains. For example, Zipcar calculated that "*after joining Zipcar, 90% of our members drove 5,500 miles of less per year*" and that "*each and every Zipcar takes 15 personally-owned vehicles off the road*"

(Zipcar, 2015).

To achieve operational efficiency, sustainability, and cost effectiveness, companies and non-profit organizations face significant planning and operational challenges as they expand carsharing programs in North America. The main issue is that, while customers favor one-way rental option that provides convenience for trip planning, meeting one-way demand can be costly, as they often cause imbalanced supply-demand in different sharing stations, which could lead to additional expense for car relocation. Another important problem is to decide where to locate carsharing stations, and how many cars should be placed at each station. The Carsharing often works the best in neighborhoods with low vehicle ownership rates outside North America, but does not necessarily reduce the total CO_2 emissions from transportation (Martin and Shaheen, 2011). The location design problem could be further complicated by having mixed types of cars for fulfilling diverse customer demand, while each type has their individual purchase cost and emission performance.

4.1.1 Focus of the Chapter and Contributions

In this chapter, we consider carsharing fleet location design and relocation in a metropolitan area, which we divide into multiple zones with one-way and round-trip demands in between pairs of zones. We model the demands based on discrete time periods, and each zone has a contracted base location to hold cars. We optimize the locations for a fleet of mixed cars, and decide how many cars of each type should be placed at each location. Constructing a spatial-temporal network in which each node represents a location at a discrete time period, we utilize the classic minimum-cost flow formulation (see Ahuja et al., 1993) to model car movement for satisfying demands over finite time periods, which can be seen as a finite horizon for repeatedly running our model and implementing its solutions. We develop two integer linear programming (ILP) formulations for modeling the problem under different assumptions. Both formulations are parameterized by demand and cost parameters, with integer decision variables representing the number of cars at each location, and

also car movements including rental operations and relocation.

An important contribution of the work resides in the utilization of a spatial-temporal network to model car movements, allowing to construct and optimize the ILP models efficiently. Later, we test diverse instances generated based on the Zipcar demand and operational data in the Great Boston area, which is particularly suited to the spatial-temporal network, given that we can track the fleet's usage at every ZIP Code and hour. This allows for a clearer picture of the overall revenue, cost, and demand fulfilled at every period. Zipcar, or any other carsharing organization, can modify our models according to their own business standards and goals. For example, one can set a constraint to force both one-way and round-trip demand to be met 90% or higher for every period in each zone, to enhance the quality of service of a carsharing system. Also, limited parking spots can be added to restrict the number of cars in each location at every period. We believe that the spatial-temporal network will prove especially useful as more mathematical optimization techniques are applied to carsharing.

Through optimizing car fleet location design and relocation, the goal of this work is to investigate the impact of having different types of cars in a carsharing program, and whether a mixed car fleet can (a) increase revenue and/or (b) limit the overall CO_2 emissions from car usage. Different carsharing programs have moved toward increasing the percentage of electric vehicles (EVs) and alternative fuel vehicles in their fleet (see, e.g., Car2Go, 2015; City Carshare, 2014; Zipcar, 2015). We will validate our results by testing instances generated based on real Zipcar data in the Greater Boston area to see whether demand for EVs or other fuel-efficient cars can gain sufficient revenue, despite higher purchase and maintenance costs of those vehicles. We will show that having a more diverse car fleet, which meanwhile makes carsharing programs more appealing to non-traditional customers, may actually have a greater environmental impact.

With increasing concerns on global climate change, it is expected that legislation will be proposed to control greenhouse gas emissions by limiting companies CO_2 emission

(Absi et al., 2013). Although carsharing companies' impact on CO_2 emission reduction has been documented, there have been very few studies on the trade-offs between CO_2 emission reduction and carsharing revenue using mathematical optimization approaches. In this chapter, we consider a mixed fleet of EVs, plug-in hybrid electric vehicles (PHEVs), hybrid cars, and regular cars, due to the fact that the EVs offered by Zipcar (or other carsharing companies) often hold a 35-mile maximum travel range. In addition, EVs are often offered in the form of the Honda Fit EV or similar vehicles, and for many families, having only EVs as their car rental choice is a big deterrent. Using a mixed car fleet, we can explore the potentials of accessing untapped, non-traditional customer markets, while achieving the goal of lowering CO_2 emissions.

We note that the work discussed in this chapter has been published in Chang et al. (2017).

4.1.2 Organization

The remainder of the chapter is organized as follows. Section 4.2 reviews the most relevant literature in carsharing, facility location, and network optimization. Section 4.3 formulates two ILP models for optimizing locations of shared cars and their relocation, subject to budget and CO_2 emission constraint. Section 4.4 analyzes the 2014 Zipcar data in the Greater Boston area, and describes the design of our experiments. Section 4.5 presents the computational and sensitivity results to demonstrate the managerial insights for carsharing under different assumptions and parameter settings. Section 4.6 concludes the chapter and states future research directions.

4.2 Literature Review

We focus on the literature of using mathematical models and optimization for carsharing system design and operations. We also review the traditional facility location models,

and discuss the ones most relevant to our ILP models in Section 4.3. Different from the traditional facility location problem, in this chapter we optimize a more complex, time-based network flow model on spatial-temporal networks. In addition to the planning of locations, we optimize car flows and relocation on the operational level.

Carsharing Service Design and Operations Given that carsharing is a fairly recent phenomenon, mathematical optimization of carsharing service is a still growing field. In the existing literature, different optimization or simulation models have been proposed to address operational problems related to various aspects of carsharing business. For example, Nourinejad and Roorda (2014) develop a dynamic optimization-simulation model to show that increasing reservation time (i.e., the time between customer requesting and picking up a car) can reduce fleet size for satisfying carsharing demand. Martinez et al. (2012), de Almeida Correia and Antunes (2012) are among the first to use integer programming models to optimize depot locations, however, for bike sharing rather than carsharing in the City of Lisbon. Nair and Miller-Hooks (2014) optimize locations of carsharing stations, their capacities, and car inventories via solving a bilevel integer programming model, to establish supply-demand equilibrium in carsharing network design. Boyacı et al. (2015) consider a pure EV fleet for carsharing, and decide its optimal fleet size and locations. Recently, He et al. (2016) investigate a planning problem for EV carsharing service providers to choose service regions and allocate EVs, given fixed locations of charging stations. They combine both customer behavior analysis and optimization for managing imbalanced demand patterns across different regions.

In this chapter, we design the location of a carsharing fleet with mixed types of cars and optimize their operations under both one-way and round-trip demands. Indeed, although allowing one-way rentals may improve demand coverage, it brings significant operational challenges due to possible demand-supply imbalance. Therefore, the existing literature has focused on how to mitigate this imbalance for systems with one-way carshares. Febbraro

et al. (2012) utilize a rolling horizon framework to derive real-time relocation policy for carsharing. Weikl and Bogenberger (2013) explore relocation strategies for free-floating carsharing systems given static demand. They develop an integrated two-step model for optimal car positioning and then relocation. Both Nair and Miller-Hooks (2011) and Barrios and Godier (2014) use optimization tools to investigate the trade-offs between changing fleet sizes and hiring car redistributors to maximize demand coverage and minimize the supply-demand imbalance in different car sharing stations.

To model car flows that satisfy one-way and/or round-trip demand, a direct method is to use a spatial-temporal network constructed based on time-varying demand between origins and destinations. We refer the interested readers to de Almeida Correia and Antunes (2012); Fan (2014); Kek et al. (2009) as representative work that have developed various spatial-temporal networks for determining parking locations and designing relocation strategies. In Section 4.3, we will describe the details of our design of the spatial-temporal network. Differently from the existing work, we develop constraints with auxiliary binary integer variables for preventing denied trips that could appear if using a generic spatial-temporal network. We also incorporate multiple car types and CO_2 emission limit into our ILP models.

Lastly, the carsharing services involve many sources of uncertainty. In the previously reviewed paper by He et al. (2016), the authors optimize the composition of service regions using a distributionally robust optimization approach, by assuming uncertain carsharing demand and fuel price with ambiguously known distributions. Lu et al. (2018) consider a carshare fleet allocation problem under random one-way and round-trip demands. They optimize a two-stage stochastic program to minimize the total costs of car allocation and parking lots/permits purchased for reservation-based or fleet-float carsharing systems, while penalizing the unfulfilled amount of random demand. They employ a branch-and-cut algorithm for optimizing the large-scale scenario-based integer programming reformulation modeled via the Sample Average Approximation approach.

Facility Location and Relations The location design and relocation model we consider in this chapter also has close relations with the facility location problem, but generalizes the classic models to a large extent by incorporating time-based demand and the resulting spatial-temporal operations of shared cars for matching demand. We refer to Daskin (2011); Kariv and Hakimi (1979); Malandraki and Daskin (1992) which discuss the classic facility location problems based on covering, p-median, p-center models and their extensions, which consider static, aggregated demand varying in location but not in time. In the past decades, the facility location problem has been incorporated or extended for network design, transportation planning, joint location-inventory control, and supply chain management (see, e.g., Daskin et al., 2002, 2005; Maass et al., 2016; Magnanti and Wong, 1984; Melkote and Daskin, 2001; Melo et al., 2009; Owen and Daskin, 1998; Shen et al., 2003, 2011; Snyder et al., 2007). Snyder (2006) provide a comprehensive review of facility location studies under various uncertainties, e.g., supply, demand, and network topologies.

4.3 Problem Description and Modeling

In this section, we describe the notation, and illustrate the construction of a spatial-temporal network according to one-way and round-trip demand. We can then formulate two ILP models, in which we embed a minimum-cost flow model based on the spatial-temporal network to model car movements and relocation.

4.3.1 Formulation of Spatial-Temporal Network

We divide a metropolitan area into zones, and denote the set of all zones by I. Each zone has a contracted base location for parking mixed vehicle fleet to satisfy carsharing demands over T periods. Given demand data, we characterize the travels between origin-destination zones, and let $O \subseteq \{0, 1, \ldots, T\}$ and $D \subseteq \{0, 1, \ldots, T\}$ be the sets of starting and ending periods, respectively. We consider J types of cars, varying in purchase cost, demand pro-

portion, per mile revenue, maintenance cost, and CO_2 emissions per mile driven. Let $d_{ii'jts}$ and $r_{ii'jts}$ be the demand and unit revenue for renting a type j car starting from zone i at period t and returning to zone i' at period s, for all $j \in J$, $i,\ i' \in I,\ t \in O$, and $s \in D$, respectively, and if $i' = i$ (i.e., a round-trip rental), we omit the index i' and use d_{ijts} and r_{ijts} for simplicity. Let c_j be the per period CO_2 emissions of a type j car, for all $j \in J$. Let b_{jt} and p_{jt} be the type j car's maintenance cost and idle cost during period t, respectively. Let m_j be the cost of purchasing a type j car, for all $j \in J$. We have a budget limit $\mathcal{F}$ for purchasing all the cars and a limit $\mathcal{H}$ for restricting the total CO_2 emissions generated over the T periods. Therefore, $v_j^{\max} = \lfloor \frac{\mathcal{F}}{m_j} \rfloor$ is the maximum number of type j cars available to be located in all the zones. We use $l_{ii'}$ to denote the minimum traveling time between zone $i \in I$ and zone $i' \in I$ and c^{rel} to denote relocation cost per period. (Without loss of generality, the unit relocation cost c^{rel} does not depend on the car type, specific period, or the origin-destination pair of the relocation trip.)

We construct a spatial-temporal network $G(N, A)$, with each node $n_{it} \in N$ representing zone $i \in I$ at period $t \in \{0, 1, 2, \ldots, T\}$. The arcs in this network are directed and represent a spatial-temporal movement of cars from one zone to another from an earlier period to a later one. In particular, we create four types of arcs: idle arcs, one-way arcs, round-trip arcs and relocate arcs. Idle arcs, $a = (n_{it}, n_{i,t+1}) \in A^I$, carry the flow of cars that stay in the same zone i from period t to period $t + 1$; one-way arcs, $a = (n_{it}, n_{js}) \in A^O$, and round-trip arcs, $a = (n_{it}, n_{is}) \in A^R$, carry car flows for satisfying their respective rentals traveling from zone i at period t and arriving at zone j at period s (where $j = i$ for round-trip demand); and relocate arcs, $a = (n_{it}, n_{i',t+l_{ii'}}) \in A^{REL}$, carry relocated car flows from zone i at period t to a different zone i' at period $t + l_{ii'}$.

Overall, we have the arc set $A = A^R \cup A^I \cup A^O \cup A^{REL}$ in the spatial-temporal network $G(N, A)$. We transform the various costs mentioned above into unit flow cost of each arc $a \in A$ for car type $j \in J$ as follows. First, no matter whether a car sits idle or is in use, there is a cost for maintenance and insurance. There are revenue gain and CO_2 emissions

for cars moving on one-way or on round-trip arcs. We also penalize cars moving on each idle arc, to encourage more usage, and this can be also interpreted as, e.g., parking cost. For each car of type j, we denote arc capacity, arc CO_2 emission, and arc revenue as u_{aj}, e_{aj}, k_{aj}, respectively, for all $a \in A$. The values of u_{aj} and e_{aj} are given in Table 4.1, and Table 4.2 presents the detailed cost composition for calculating k_{aj} for each arc $a \in A$ and car type $j \in J$.

Table 4.1: Arc capacity u_{aj} and unit CO_2 emission e_{aj} for vehicle type $j \in J$

Type of Arc	Capacity u_{aj}	CO_2 emission e_{aj}
Idle arcs $a = (n_{it}, n_{i,t+1}) \in A^I$	$v_j^{\max}$	0
Round-trip arc $a = (n_{it}, n_{is}) \in A^R$	d_{ijts}	$c_j(s-t)$
One-way arc $a = (n_{it}, n_{i's}) \in A^O$	$d_{ii'jts}$	$c_j(s-t)$
Relocation arc $a = (n_{it}, n_{i',t+l_{ii'}}) \in A^{REL}$	$v_j^{\max}$	$c_j l_{ii'}$

Table 4.2: Maintenance cost, idle cost, and revenue to compute arc revenue k_{aj}

Type of Arc	Maintenance	Idle	Relocation	Profit	Arc Revenue k_{aj}
Idle arc $a = (n_{it}, n_{i,t+1}) \in A^I$	b_{jt}	p_{jt}	0	0	$-(b_{jt} + p_{jt})$
Round-trip arc $a = (n_{it}, n_{is}) \in A^R$	$\sum_{\ell=t}^{s-1} b_{j\ell}$	0	0	r_{ijts}	$r_{ijts} - \sum_{\ell=t}^{s-1} b_{j\ell}$
One-way arc $a = (n_{it}, n_{i's}) \in A^O$	$\sum_{\ell=t}^{s-1} b_{j\ell}$	0	0	$r_{ii'jts}$	$r_{ii'jts} - \sum_{\ell=t}^{s-1} b_{j\ell}$
Relocation arc $a = (n_{it}, n_{i',t+l_{ii'}}) \in A^{REL}$	$\sum_{\ell=t}^{t+l_{ii'}-1} b_{j\ell}$	0	$c^{\text{rel}} l_{ii'}$	0	$-c^{\text{rel}} l_{ii'} - \sum_{\ell=t}^{t+l_{ii'}-1} b_{j\ell}$

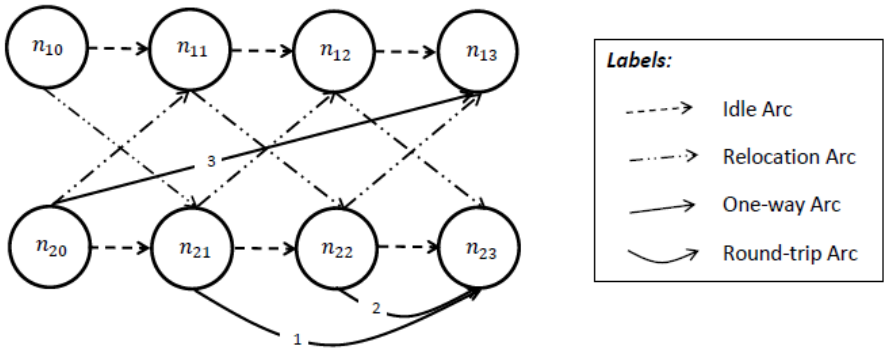


Figure 4.1: Example of a spatial-temporal network

Figure 4.1 shows an example of the spatial-temporal network, for which the problem has two zones $\{1, 2\}$, and four periods $\{0, 1, 2, 3\}$. Each node n_{it} represents zone i at period t. We use different arrows to represent the four types of arcs as stated in legend of

Figure 4.1. Specifically, one car travels on a round-trip arc (n_{21}, n_{23}), meaning that one car departs from zone 2 at period 1 and returns to the same zone at period 3; two cars travel on a round-trip arc (n_{22}, n_{23}), meaning that two cars depart from zone 2 at period 2 and return to the same zone at period 3; three cars travel on a one-way arc (n_{20}, n_{13}), meaning that three cars leave zone 2 at period 0 and return to zone 1 at period 3. The spatial-temporal network flow model allows us to track the cars' status (whether in use or being idle) in each zone from period to period.

4.3.2 A Mathematical Optimization Model

We define integer variables $x_{ij} \in \mathbb{Z}_+$ as the initial number of cars of each type $j \in J$ located in zone $i \in I$ at time $t = 0$. For each arc $a \in A$ and vehicle type $j \in J$, we define an integer variable $y_{aj} \in \mathbb{Z}_+$ as the number of cars of type j flowing on arc a. Let $\delta^+(n_{it})$ and $\delta^-(n_{it})$ denote the sets of arcs for which n_{it} is the origin node and the destination node in the network $G(N, A)$, respectively. We formulate below an integer linear program to optimize the car fleet location, size, and types, as well as their rental operations and relocation.

$$\text{(M1)} \quad \max \quad \sum_{a \in A} \sum_{j \in J} k_{aj} y_{aj} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(n_{it})} y_{aj} - \sum_{a \in \delta^-(n_{it})} y_{aj} = \begin{cases} x_{ij} & \text{if } t = 0 \\ 0 & \text{if } t \in \{1, \ldots, T-1\} \end{cases} \qquad \forall n_{it} \in N,\ j \in J \tag{4.2}$$

$$\sum_{a \in A} \sum_{j \in J} e_{aj} y_{aj} \leq \mathcal{H} \tag{4.3}$$

$$\sum_{i \in I} \sum_{j \in J} m_j x_{ij} \leq \mathcal{F} \tag{4.4}$$

$$y_{aj} \leq u_{aj} \qquad \forall a \in A,\ j \in J \tag{4.5}$$

$$x_{ij} \in \mathbb{Z}_+, y_{aj} \in \mathbb{Z}_+ \qquad \forall a \in A,\ j \in J \tag{4.6}$$

The objective function (4.1) maximizes the total revenue of operating cars purchased for satisfying carshare demands over the T periods. The flow balance constraints (4.2) ensure

that the number of cars leaving each node is equal to the number of cars entering the same node for periods that are not 0 or T, for each car type. At period 0, x_{ij} cars of type j are available in zone i. Constraint (4.3) limits the amount of CO_2 emissions generated over the T-period time span. Constraint (4.4) reflects the total budget limit of purchasing cars. Constraints (4.5) ensure that flows on each arc do not exceed the corresponding arc capacity (i.e., demand values for one-way and round-trip arcs).

4.3.3 Model Justification and Assumptions

We justify the validity of the above model as follows. Without loss of generality, we assume positive revenue on arcs $a \in A^R \cup A^O$ (i.e., $r_{ijts} - \sum_{\ell=t}^{s-1} b_{j\ell} > 0$ for all $a \in A^R$, and $r_{ii'jts} - \sum_{\ell=t}^{s-1} b_{j\ell} > 0$ for all $a \in A^O$ in Table 4.2). The objective function (4.1) intends to increase y_{aj} on arcs $a \in A^R \cup A^O$ and decrease y_{aj} on arcs $a \in A^I \cup A^{REL}$ (given positive cost k_{aj} on any arc $a \in A^I \cup A^{REL}$ in Table 4.2), while constraints (4.5) bound the values of y_{aj} on each arc $a \in A$ and car type $j \in J$ from above. In particular, $y_{aj} \leq u_{aj}$ for any $a \in A^R \cup A^O$, where u_{aj} is the carsharing demand on arc a for type j cars. Thus, if possible, the flow y_{aj} on arc $a \in A^O \cup A^R$ will be equal to the corresponding demand (i.e., arc capacity) driven by revenue; excesses of cars of each type (if exist) will flow on the arc $a \in A^I \cup A^{REL}$ to either stay idle or be relocated.

We assume that demand losses in each zone $i \in I$ for each car type $j \in J$ at period $t \in \{0, 1, \ldots, T\}$ will immediately disappear and will not be rolled over to future periods. This is consistent with the real-world situation, in which customers will seek alternative transportation forms if no cars are available at nearby locations. For simplicity, we also assume no car substitutions between different types. Also, to maintain flow balance, we assume that cars will be always returned on time. In reality, carsharing companies charge hefty fines for not returning cars on time, and there would be legal ramifications if a car were not returned at all.

4.3.4 Model Extension and Modification

Our study can be extended in the following ways. First, we can modify the capacities of idle arcs $(n_{it}, n_{i,t+1})$ in zone i for each period t by letting them be the number of parking spots reserved in zone i. Such parking availability can be given as input parameter, or we can define the number of parking spots reserved in each zone as integer variables, in addition to the current variables x and y. We can then optimize the total revenue offset by fees paid to reserve parking. Both modifications do not change the complexity of our integer programing models.

Second, our model can handle first-come, first-served (FCFS) principle of a carsharing system by adding constraints to (M1). FCFS is a service policy to serve carsharing requests in the order they arrive, without other preferences. In our problem context, the FCFS principle requires an idle arc $(n_{it}, n_{i,t+1})$ having positive flows only if all the demands, both one-way and round-trip, have been fulfilled in zone i at period t. To model this rule, we introduce new binary variables $z_{it}^j \in \{0, 1\}$ for each zone i, car type j, and period t and let

$$z_{it}^j = \begin{cases} 1 & \text{if any cars of type } j \text{ being idle in zone } i \text{ at period } t \\ 0 & \text{otherwise.} \end{cases}$$

We add the following constraints to (M1) to enforce FCFS and ensure zero denied trips:

$$\sum_{a \in \delta^+(n_{it}) \cap (A^O \cup A^R)} (u_{aj} - y_{aj}) \leq v_j^{\max}(1 - z_{it}^j) \qquad \forall i \in I,\ t = 0, 1, \ldots, T-1,\ j \in J \tag{4.7}$$

$$y_{(n_{it}, n_{i,t+1}),j} \leq v_j^{\max} z_{it}^j \qquad \forall i \in I, t = 0, 1, \ldots, T-1,\ j \in J \tag{4.8}$$

$$z_{it}^j \in \{0, 1\} \qquad \forall i \in I,\ t = 0, 1, \ldots, T-1,\ j \in J \tag{4.9}$$

According to the above constraints, when some cars of type j stay idle at node n_{it}, we have $z_{it}^j = 1$ by constraints (4.8), which will enforce the right-hand side of the corresponding constraint (4.7) being 0, indicating that all the demands for cars of type j have been fulfilled in zone i at period t, i.e., $y_{aj} = u_{aj}$ for all one-way and round-trip arcs emanating from

node n_{it}, such that $a \in \delta^+(n_{it}) \cap (A^O \cup A^R)$. On the other hand, if any demand in zone i at period t for type j cars has not been fulfilled, i.e., there is a one-way or round-trip arc emanating from node n_{it} carrying flows that have not reached the arc capacity, constraints (4.7) imply that $z_{it}^j = 0$. Then constraints (4.8) ensure no cars of type j being idle in zone i at period t.

An Illustration: We illustrate how constraints (4.7)–(4.9) can successfully enforce the FCFS principle by constructing an example based on Figure 4.1. Suppose that $y_{(n_{21},n_{22}),1} > 0$, which indicates some cars of type 1 being idle in zone 2 from period 1 to period 2. Then constraints (4.8) imply that $z_{21}^1 = 1$ and thus we have $y_{(n_{21},n_{23}),1} = u_{(n_{21},n_{23}),1}$, i.e., all carsharing demands are fulfilled in zone 2 at period 1 for type 1 cars according to constraints (4.7). On the other hand, suppose that $y_{(n_{21},n_{23}),1} < u_{(n_{21},n_{23}),1}$, which indicates that some demands for type 1 cars are not fulfilled on node n_{21}; then, constraints (4.7) enforce $z_{21}^1 = 0$ and thus $y_{(n_{21},n_{22}),1} = 0$ due to constraints (4.8), i.e., no cars of type 1 remain idle in zone 2 from period 1 to period 2.

In this chapter, we also study a more restrictive integer linear program than (M1):

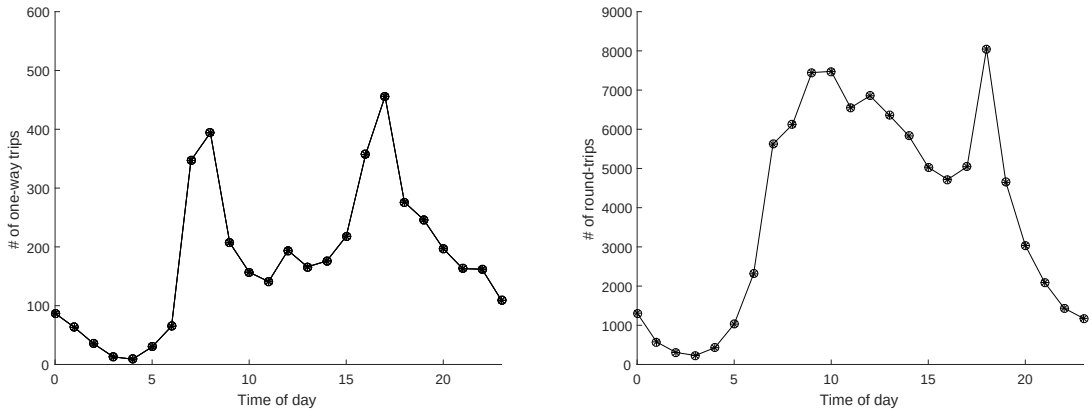
$$\begin{aligned} \text{(M2)} \quad \max \quad & \sum_{a \in A} \sum_{j \in J} k_{aj} y_{aj} \\ \text{s.t.} \quad & \text{(4.2)–(4.9)} \end{aligned} \tag{4.10}$$

4.4 Boston Zipcar Data and Experimental Design

The City of Boston's carsharing program, partnered with Zipcar, has been in existence since 2000. We utilize their demand and operational data collected in 2014 to generate instances for our numerical studies.

4.4.1 Data Descriptions

The dataset[1] includes every Zipcar reservation made in the Greater Boston area between October 1st and November 30th in 2014. The data records show the number of reservations started at each hour of a 24-hour day, as well as the ZIP code for where the trip began. Given this information, we can reasonably assess the average number of trips made at each hour of the day at any given ZIP code. We plot the average number of one-way and round-trip weekday demands in Figure 4.2. Note that trip reservations significantly increase from 7am to 9am. Demand slows down throughout the afternoon, but jumps back up at 5pm. Following Figure 4.2, we divide a day into two parts: peak hours, from 7am to 7pm, and off-peak hours, from 7pm to 7am of the next day. We solve our models over a 12-hour horizon by using the data of peak-hour demand (which accounts for the majority of whole-day demand), with each period equal to one hour.



(a) One-way demand pattern (b) Round-trip demand pattern

Figure 4.2: Time-based one-way and round-trip demands in the Zipcar Boston data

We partition the Greater Boston area into 60 different zones, one ZIP code for each zone, shown in Figure 4.3. The number of one-way and round-trip rentals with their origin

[1] https://data.cityofboston.gov/Transportation/Zipcar-Boston-Reservations/863f-ps42. We choose this data because the carsharing demands in Boston have been relatively stabilized by this time, fourteen years into the program. Moreover, this is the only dataset open to the public from Zipcar including both one-way and round-trip reservations.

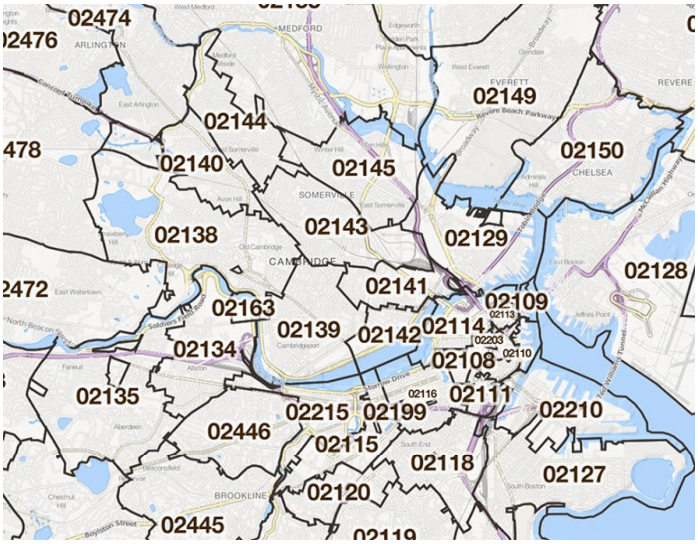


Figure 4.3: Zip code map (zones) of the Greater Boston area

and destination zones, and also the start time of each rental are provided in the raw data.

4.4.2 Experimental Design

In all our instances, we consider four types of cars, all of which currently are or have been a part of Zipcar fleets: the 2014 Honda Fit Electric Vehicle (EV), 2015 Honda Fit LX (LX), the 2014 Honda Accord Plug-in Hybrid Electric Vehicle (PHEV), and the 2015 Honda Accord Hybrid (Hybrid). EVs rely solely on an electric motor, whereas PHEVs and Hybrids stand in the middle between EVs and traditional gas-only vehicles like the LX. PHEVs and Hybrids utilize both gasoline and electricity, but produce fewer CO_2 emissions than gas-only vehicles. Since PHEVs and Hybrids also utilize a gasoline engine, they does not suffer the same distance limitations as EVs do. We foresee the PHEVs and Hybrids being viable alternatives for customers who dislike the limitations of EVs.

In the original dataset, the one-way reservations are far fewer than the round-trip ones. This is mainly because one-way carsharing was a relatively new service and has a relatively higher rental price as compared to the traditional round-trip carsharing. In our tests, we generate based on the original dataset more instances with four different one-way pro-

portions, being 0%, 40%, 80% and 100% of the total carsharing demands, to study how different one-way to round-trip demand ratios affect carsharing operations and results. To do this, first, for each starting zone and time period from 7am to 6pm, we compute the average number of round-trips in the original demand data. We split certain number of round-trips, according to the desired percentage of one-way trips, into two one-way trips, and randomly pick an inserted zone for each split (as the destination of the first trip and the origin of the second trip) according to the probabilities of having one-way trips between zone pairs observed in the original data.

We assume that, given that a Zipcar EV only has a 35-mile travel range, the durations of all the one-way and round-trip rentals are 1, 2, 3, or 4 hours (periods), with equal probability. We also assume 40%, 20%, 20%, 20% of the total carsharing demands for LXs, EVs, PHEVs, and Hybrids, respectively, the same for both one-way and round-trip rentals. We construct the spatial-temporal network based on the 60 zones and the 12-hour operational horizon, which consists of 720 nodes and 1,097,040 arcs.

For the two ILP models (M1) and (M2), we use $r_j = \$7.75$ as the round-trip revenue per hour and $r'_j = \$12$ as the one-way revenue per hour for all car types $j \in J$, which are the rental rates used by Zipcar in Boston. We calculate $r_{ijts} = r_j(s-t)$ and $r_{ii'jts} = r'_j(s-t)$ for round-trip and one-way arcs, respectively, for all $i,\ i' \in I$, $s \in D$, and $t \in O$. We use $b_{jt} = 0$ since the maintain cost is negligible over a 12-hour horizon, and set $p = p_{jt} = \$0.4$ for all periods t and car types j (Chesto, 2015). Assuming that each customer drives 10 miles per hour on average, we set the CO_2 emission c_j for each car type j per period as the numbers detailed in Table 4.3 below, based on the per mile CO_2 emissions of the four types of cars estimated by the U.S. Department of Energy[2]. We use the relocation cost $c^{\text{rel}} = \$8$ per period. Table 4.3 summarizes the values of parameters $m_j,\ r_j,\ b_j,\ p,\ c^{\text{rel}},\ c_j$ for the four types of cars used in our computation.

[2]http://www.fueleconomy.gov/

Table 4.3: MSRP, revenue, maintenance cost, penalty cost, and emission per vehicle type

Vehicle type j	MSRP (m_j)	Revenue (r_j/r'_j)	Relocation (c^{rel})	Idle (p)	CO_2 (c_j)
Fit EV 2014	\$37,445	\$7.75/\$12.00	\$8.00	\$0.4	1200 g
Fit LX 2015	\$17,270	\$7.75/\$12.00	\$8.00	\$0.4	2960 g
Accord PHEV 2014	\$39,780	\$7.75/\$12.00	\$8.00	\$0.4	2000 g
Accord Hybrid 2015	\$29,305	\$7.75/\$12.00	\$8.00	\$0.4	2270 g

All the instances are solved using Python 2.7 to call the solver Gurobi 6.0.3 for directly optimizing ILP models. All programs are run on Microsoft Windows 8.1 64-bit operating system on a Lenovo Laptop with Intel(R) Core(TM) i5-6200U CPU 2.30 GHz and 8.0 GB RAM.

4.5 Computational Studies and Analysis

We test a diverse set of instances to determine the effects of CO_2 emission and budget constraints on revenue and the quality of service, measured by the rates of demand fulfillment and denied trips. We report the CPU time, optimal objective values, and service quality results to compare (M1) and (M2). We also analyze the optimal design of shared car fleets, and conduct sensitivity analysis by varying parameter values.

4.5.1 CPU Time and Objective Value

Tables 4.4 and 4.5 present the CPU time (in seconds) and the optimal objective values of instances solved by models (M1) and (M2), respectively. For all the instances, we set the budget $\mathcal{F} = \$10$ million for purchasing cars and the CO_2 emission limit $\mathcal{H} = 5 \times 10^6$ grams.

We set the CPU time limit as 20 minutes for computing all the instances. According to Tables 4.4 and 4.5, only the (M1) model with 0% one-way demand exceeds the time

Table 4.4: CPU seconds and optimal objective value ($) of model (M1)

One-Way Proportion	**0%**	**40%**	**80%**	**100%**
CPU Time	61.44*	39.39	52.77	63.59
Optimal Objective Value	$16,709.95	$17,382.00	$20,779.70	$21,732.05

*: the best optimality gap $= 0.031\%$ is achieved.

Table 4.5: CPU seconds and optimal objective value ($) of model (M2)

One-Way Proportion	**0%**	**40%**	**80%**	**100%**
CPU Time	169.33	233.55	998.31	91.54
Optimal Objective Value	$16,701.15	$17,371.20	$20,732.25	$21,732.05

limit, but achieves 0.031% optimality gap very quickly, after 61.44 seconds. The average CPU time for running (M1) is 54.30 seconds, as compared to the average CPU time being 723.43 seconds for running (M2) with the additional constraints (4.7)–(4.9) for imposing the FCFS principle.

As the proportion of one-way trips increases, the optimal objective values of both models increase (approximately by 30% from 0% to 100% one-way proportions). We note that the optimal objective values of (M1) and (M2) are very similar, but the CPU time increases significantly from (M1) to (M2). This is because (M2) includes the additional "Big-M" constraints (4.7)–(4.9), which are recognized as being very inefficient for solving integer programs. Later, we will report the percentages of denied trips to measure the number of trips that violate the FCFS principle when using (M1) without constraints (4.7)–(4.9). (Note that feasible solutions to (M2) always yield zero denied trips.) The model (M1) automatically yields very low amount of denied trips (less than 1% of the total demand) even without the FCFS constraints. Thus, we recommend using (M1) for carsharing practitioners to achieve better trade-offs between the computational efficiency and model complexity. In the rest of the chapter, we only solve (M1) for all the remaining instances.

4.5.2 Quality of Service

To evaluate the quality of service (QoS) for our carsharing model, we study several system metrics, including the percentage of unfulfilled demands, the percentage of denied trips, idle car hours, and car utilization rates. Here, denied trips refer to the number of unfulfilled demands when there are idle cars to serve them. In other words, the amount of denied trips reflects the magnitude of violation of the FCFS principle. Table 4.6 displays the QoS metrics as one-way proportion varies. We continue using $\mathcal{F} = \$10$ million and $\mathcal{H} = 5 \times 10^6$ grams.

Table 4.6: Quality of Service Metrics

One-Way Proportion	Unfulfilled Rentals	Unfulfilled Rentals (%)	Denied Trip (%)	Idle Vehicle Hours	Vehicle Utilization
0%	164	15.43%	1.03%	1412	61.22%
40%	53	5.74%	0.76%	1690	56.48%
80%	44	4.84%	1.65%	1667	56.82%
100%	10	1.14%	0.11%	1653	57.43%

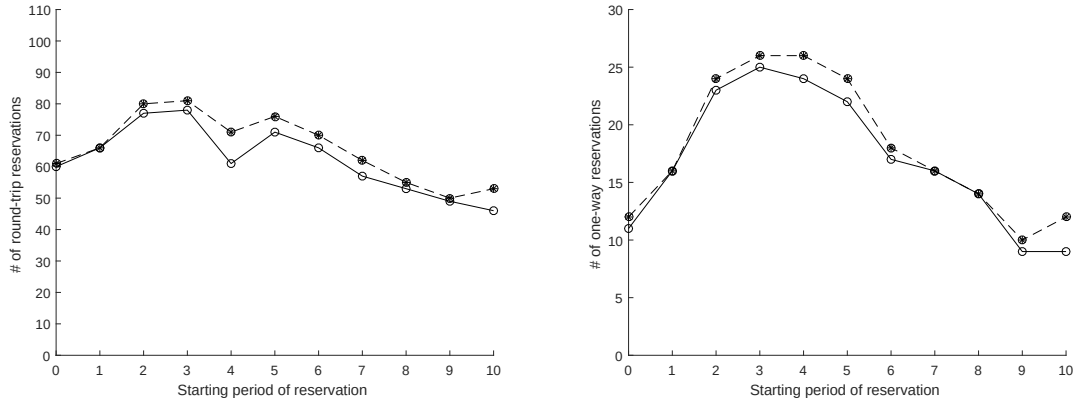
The unfulfilled rental metric is largely dependent on budget and emission constraints, but provides a useful view for whether our model is capable of fulfilling demand. The number of unfulfilled rentals is high for the 0% one-way case, but the system also has the highest car utilization. Under other one-way proportion settings, we have much fewer unfulfilled demand rates, ranging from 1.14% to 5.74%.

The model (M1) is also capable of enforcing the FCFS principle without explicitly creating constraints (4.7)–(4.9) as in model (M2). FCFS is a vital business principle for carsharing companies, since a primary business objective is to provide a reliable service and available cars to customers as they arrive in the system. If customers are turned away when cars are available, or the system rejects reservations because it forecasts more profitable trips in the future, carsharing companies may lose a competitive business edge. In Table 4.6, the percentages of denied trips are extremely low, ranging from 0.1% to 1%. In other words, demands are fulfilled between 99% and 99.9% of the time if the corresponding type

of cars are available.

We use the idle car hours to calculate the car utilization rates, which are on average over 55% for each car. Overly high utilization rates will cause lowered QoS metrics, while low utilization rates mean that cars are unnecessarily purchased under too loose of a budget. In general, the higher the utilization rate is, the higher the number of unfulfilled rentals will be. In our tests, we use a relatively low penalty for cars being idle, which allows high rental fulfillment. Setting higher penalty costs can ensure that idle car hours decrease and car utilization rates increase.

Recall that under 40% one-way proportion, the demand fulfillment rate is on average 94%. We show in Figure 4.4 the number of fulfilled one-way and round-trip reservations based on their starting periods. The dashed line represents carsharing demand (i.e., capacity u_{aj}), while the solid line is the number of trips satisfied starting from each period. Note that trip fulfillment rate is low in the last period, since trips started in the last period must return within one period to satisfy flow balance. Essentially, trips made in the last hour are one-hour trips only and the system tends to fulfill other trips instead. Otherwise, trip fulfillment rate is extremely high in each zone throughout the 12-hour horizon.



(a) Round-trip (b) One-way

Figure 4.4: Number of fulfilled one-way and round-trip reservations

4.5.3 Effects of CO_2 Emission Limit

We vary the CO_2 emission limit to see its effect on the optimal car fleet composition. Figure 4.5 shows how the number of cars of each type in the fleet changes as the right-hand side of the CO_2 emission constraint varies. We use $\mathcal{F} = \$10$ million as the total budget for purchasing cars.

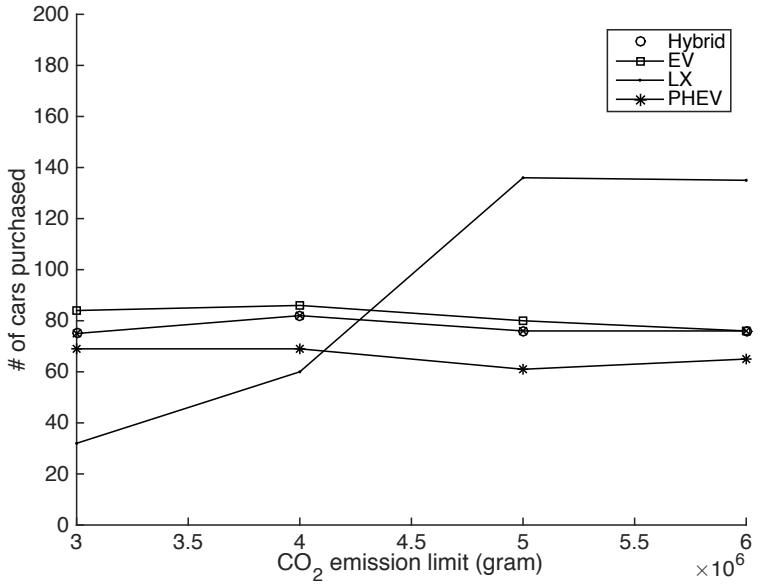


Figure 4.5: Number of cars purchased for each type given CO_2 emission limit

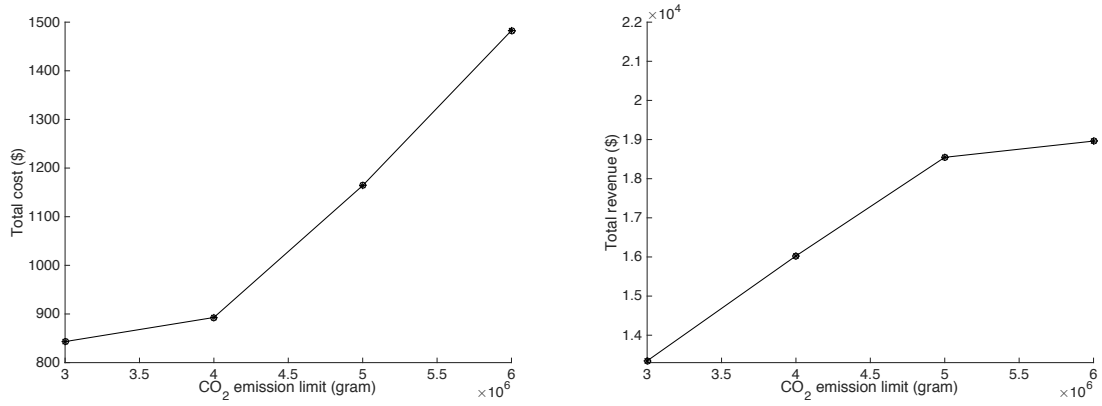
In Figure 4.5, we note that between 3×10^6 and 6×10^6 grams of CO_2 emissions allowed, EVs, Hybrids, and PHEVs comprise a majority of the fleet. Few LXs are purchased when the emission limit is low. Our model allows for a diverse fleet when restricted by the CO_2 emission limit.

As the CO_2 emission limit becomes less restrictive, the number of LXs purchased goes up, as expected. We also note that the model still shows an obvious preference for Hybrid cars, which are consistently purchased more than PHEVs. This can probably be attributed to the high cost of a PHEV, although PHEVs and Hybrids are fairly similar in technology and function. This model can be used by companies who seek to determine the compo-

sition of a large fleet of cars to purchase and use for several years. A fleet composition analysis using this model can sharply differentiate the more profitable car models while still benefiting the environment.

Many adjustments can also be made to further determine a proper fleet. For example, the current demands consist of 40% LXs, 20% EVs, 20% PHEVs, and 20% Hybrids. Given the wide range of car preferences and needs throughout carsharing communities, we can increase or decrease the demand for each vehicle type and re-solve the resulting instance of (M1). We can also, if necessary, add further constraints to ensure that a certain percentage of demand is fulfilled for some specific car type, e.g., 90% of the demand for EVs is met.

Figures 4.6a and 4.6b display the changes of the total cost and the total profit (i.e., revenue minus cost) as we increase the CO_2 emission limit.



(a) Total cost

(b) Total revenue

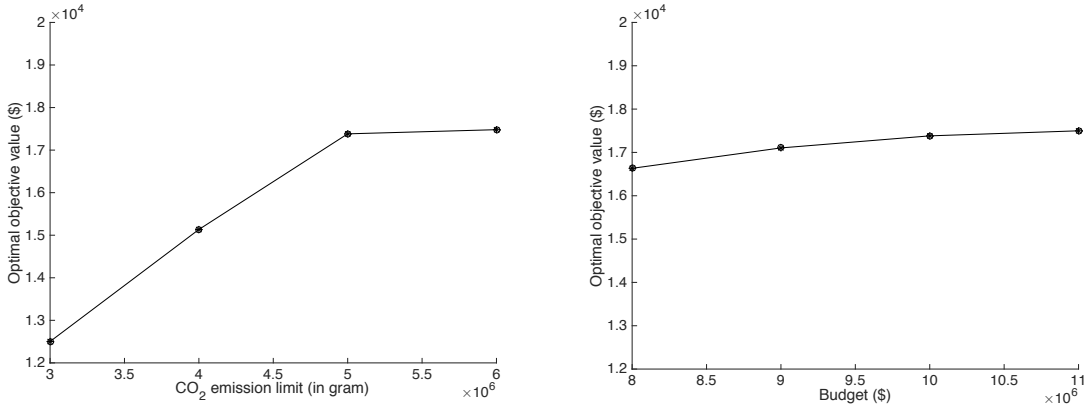
Figure 4.6: Total cost and revenue given by different CO_2 emission limit $\mathcal{H}$

In Figures 4.6a and 4.6b, as the CO_2 emission limit increases, the total revenue over the 12-hour horizon plateaus around $18,000. This is significant since imposing CO_2 emission limit on a carsharing fleet may not hurt the revenue, which will reach a threshold value at some point and the cost to purchase additional cars will not justify attempting to meet further demand. On the other hand, imposing emission limit can reduce tens of thousands of grams of emissions. For example, as we observe, if we place the CO_2 emission limit

at $\mathcal{H} = 5 \times 10^6$ grams, we would only lose \$415.25 (or 0.02%) revenue over the 12-hour horizon, as compared to the case where CO_2 emission limit is set as 6×10^6 grams.

4.5.4 Sensitivity Analysis

The primary constraints in the (M1) model concern about capacity, emission limit, and budget for purchasing cars. In this section, we vary the values of parameters in these constraints to analyze the sensitivity of our results. We fix $\mathcal{F} = \$10$ million and $\mathcal{H} = 5 \times 10^6$ grams, while varying the CO_2 emission limits and car purchasing budgets, respectively. We consider 40% one-way proportion setting, and assume that the demand profile is fixed regardless of the increase or decrease of car availability or CO_2 emission limit. Figures 4.7a and 4.7b show the sensitivity results of CO_2 emission and budget constraints on the optimal objective value of (M1), respectively.



(a) CO_2 emission limit $\mathcal{H}$ (b) Budget limit $\mathcal{F}$

Figure 4.7: Optimal objective values under varying CO_2 emission limit and budget

In Figure 4.7a, the optimal objective value begins leveling out fairly quickly at around CO_2 emission limit of 5×10^6 grams. The optimal objective value in Figure 4.7b for the budget constraint is much less sensitive, and is only increased by \$865 (as the 12-hour net revenue of running the system) from $\mathcal{F} = \$8$ million to $\mathcal{F} = \$11$ million, under the same demand profile.

We also test whether varying the percentages of each car type will affect the optimal objective value. We use the percentage numbers in Table 4.7 to generate instances with diverse demand composition for the sensitivity analysis. To generate fleet capacities, we make the percentage of LXs the biggest, followed by the ones of Hybrids, EVs, and PHEVs.

Table 4.7: Tests of different percentages of EVs, LXs, PHEVs, and Hybrids

	Test 1	Test 2	Test 3	Test 4
EV	10%	20%	20%	25%
LX	60%	50%	40%	30%
PHEV	10%	10%	20%	15%
Hybrid	20%	20%	20%	30%

In Figure 4.8, we evaluate how the optimal objective value changes as percentages of the demand for four car types vary, with $\mathcal{F} = \$10$ million and $\mathcal{H} = 5 \times 10^6$ grams. We note that the objective value increases as the demand percentage for LXs decreases and the demand percentages for other types of cars even out. However, the difference between the maximum and the minimum objective values ($17433.86 and $16297.35, respectively) is not very large, being only $1136.25. The percentages of different car-type demands, therefore, do not heavily impact the revenue.

4.5.5 Result Summary

Transportation accounts for 27% of greenhouse gas emission (or 6,673 million metric tons of CO_2) in the United Sates (see United States Environmental Protection Agency, 2013). This number could be significantly reduced through wider investment in carsharing, which has been proven to decrease car ownership rates and the related carbon footprint in the past decade. Making carsharing more successful among diverse communities requires purchasing and locating fleets that match the community's exact needs and profile. To meet this goal, we utilized the spatial-temporal network to model car rental and relocation operations corresponding to time-varying demands and we optimize carsharing fleet location and relocation with mixed car types.

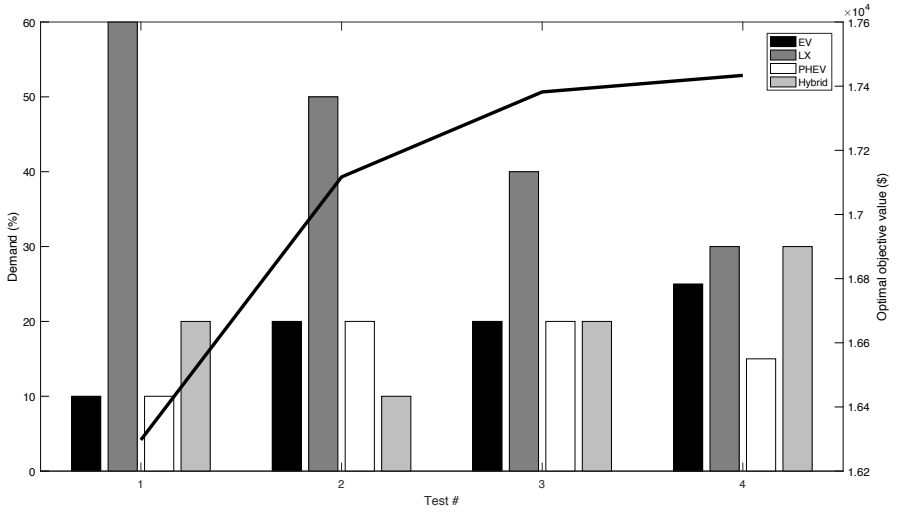


Figure 4.8: Optimal objective values of Tests 1, 2, 3, 4 in Table 4.7 with different demand compositions

We considered instances to determine the types and numbers of cars that should be purchased, as well as their locations for designing a self-sustained, efficient carsharing system. Testing various demand compositions for four different types of cars, we noted that as long as the CO_2 emission limit is low, the numbers of PHEVs, Hybrids, and EVs purchased are fairly similar. As we increase the CO_2 emission limit, the majority of cars purchased become LXs and Hybrids. Our results confirmed a clear preference for certain types of cars under different parameter settings. We demonstrated which cars will be profitable in the long run. We provided a powerful optimization tool for companies seeking to purchase a long-term fleet, given certain budget limits, revenue goals, and CO_2 emission reduction goals.

We also imposed a CO_2 emission constraint on the spatial-temporal network. Given that many carsharing companies make it a part of their company mission to reduce CO_2 emissions, we believe this will be a valuable component when deciding fleet allocation. By changing the total limit on CO_2 emissions, we found that imposing this constraint would allow a fleet to significantly reduce CO_2 emissions, while affecting only a small percentage of the total revenue.

Our approaches also worked well for fulfilling demands and preventing denied trips. We tested two models, (M1) and (M2), of which the latter explicitly enforces trips being served if cars are available in the same starting location. However, even though (M1) does not explicitly include such constraints, our results showed that it automatically enforced low denied-trip percentages (between 0.1% to 1%) in all the tested instances. Overall, demand losses are low for all the instances under different budgets and emission limits. Each car has a high utilization rate, at over 55% on average, which is much higher than the average utilization rate of private cars.

We were also able to directly view the effectiveness of the car purchasing budget. The sensitivity analysis of the budget against the revenue provided a simple but useful tool for companies to study how much they wish to spend on purchasing new cars each year. For example, in Section 4.5.2, we found that increasing the budget by $3 million only led to a $865 revenue increase over the 12-hour horizon, under the same amount of demand. Understanding the impact of the budget on revenue can lead to clearer analysis of how to increase revenue through other methods, such as locating carsharing spaces closer to public transport.

4.6 Concluding Remarks

In this chapter, we constructed a spatial-temporal network to model car movements and formulated a carsharing location design and relocation problem as integer linear programs with an embedded minimum-cost flow model. Each node in the network represents a base location at a given time from a set of discrete periods. We investigated the impacts of having different car types in a carsharing fleet and how diverse fleets could increase revenue, and meanwhile limit CO_2 emissions from car usage. We introduced two ILP models, (M1) and (M2), to ensure high QoS reflected as low unfulfilled demand rates and low denied trip rates. Using 2014 Zipcar data in the Great Boston area, we tested a diverse set of instances

with both (M1) and (M2). Our numerical experiments showed that our approach works well for fulfilling both one-way and round-trip carsharing demands while limiting the total CO_2 emissions. In addition, we maintained high car usage and low denied trip rates using the (M1) model without explicit FCFS constraints.

We discussed modifications of (M1) via the introduction of additional constraints and variables, and how they would create effective alternative models for determining the optimal location and composition of mixed car fleet, under other more complex problem settings. Constraints for meeting required demand satisfaction rates, denied trip rates, revenue gains, etc., can be easily incorporated into the ILP models, which can be solved quickly by off-the-shelf optimization solvers. The related studies will be carried out in our future research.

Another promising future research direction is to study fleet distribution of shared autonomous vehicles (SAVs). Introducing autonomous vehicles into the American roadway will come through many means, but including them in carsharing fleets will likely be one of the vital first steps. Just as we study the integration of EVs into a carsharing fleet, conducting research on the integration process and allocation of an SAV fleet is the next natural step.

CHAPTER 5

An Integrated Car-and-ride Sharing System for Mobilizing Heterogeneous Travelers with Application in Underserved Communities

5.1 Introductory Remarks

In recent years, shared mobility has shown its flexibility and strength in providing convenience for personal travel and reducing traffic congestion on public roads by reducing car ownership (Martin, 2016; Shaheen et al., 2015). With increasing concerns about climate change, congestion, and fossil fuel dependency, shared mobility attracts more attention and is undergoing a fast rise in popularity and industrial growth (Chan and Shaheen, 2012). Two main mobility-sharing forms are carsharing and ride-hailing. The former provides users with access to car rental service on an hourly basis (Millard-Ball, 2005). The latter, a service initially aiming to group travelers with common itineraries (Chan and Shaheen, 2012), has evolved rapidly with the development of smartphone technologies. Although both carsharing and ride-hailing provide useful alternatives to owning personal cars, they focus on different user groups. Carsharing users usually rent cars for running errands with multiple short stops. To use a carsharing service, users are required to meet driving eligibility requirements. Ride-hailing service targets users with short, and often one-time ride needs. It especially benefits people who are not able to drive, unfamiliar with the city transit

system, and/or commuting with poor transit access.

Carsharing business started from a station-based model in the United States in the late 1990s and individuals join a membership program by paying fixed monthly or annual fee. Service providers often ask users to reserve a vehicle in advance, pick up and return the vehicle to the same station, which is often located in a population-dense region. In recent years, carsharing evolved to allow users to pick up and drop off vehicles in different locations and some free-floating carsharing only requires their users to return vehicles to a zone in service. However, the flexibility comes at the cost of vehicle stock imbalance and drives up the price to use the service.

With the support of GPS technology and broad adoption of smartphones, ride-hailing service emerged in recent years. It allows customers to request drivers through a smartphone app. As a result, its efficiency and service quality rely on hundreds of thousands of drivers who participate in real time. Working as a driver in ride-hailing service platforms became a new career choice for many people and contributes to society employment growth. However, to become a driver, one needs to pass a series of qualifications including having his/her own vehicle to meet the model year requirement. Recently, many people have decided to rent or lease a car to drive for ride-hailing from platforms such as HyreCar. However, they are still bearing the cost of operating vehicles and uncertain demand. Clewlow and Mishra (2017) show that 30-50% of ride-hailing drivers are losing money. Furthermore, those car rental services are only provided in large metropolitan areas. Nevertheless, the ride-hailing services attract much more users than carsharing, both from the passenger side and the driver side: there are more than 250 million users globally of ride-hailing versus 5 million users of carsharing.

Both carsharing and ride-hailing services rely on several characteristics of existing transportation systems to be successful, including limited parking, limited public transportation, walkability, high population density, and mixed-use neighborhoods (see, e.g., Brook, 2004; Muheim and Reinhardt, 1999). In cities with high population density, the

waiting time for both drivers and passengers could be easily reduced in ride-hailing and thus attracts more users with high efficiency (Agatz et al., 2012; Alonso-Mora et al., 2017). In fact, both carsharing and ride-hailing services are targeting users that are young, educated, have moderate income, and live in urban areas (Smith, 2016). However, for the neighborhoods that do not have populations with these characteristics, the services are less likely to be successful, and the coverage for such neighborhoods is therefore limited.

Given the variety of limitations we listed above, in this chapter, we propose a car-and-ride sharing (CRS) system, in which carsharing and ride-hailing services are integrated and co-provided to boost the mobility of heterogeneous travelers based on their characteristics and special needs. We aim to design a financially and operationally self-sustainable system, such that users with ride-hailing demands are served by drivers with carsharing needs. We aim to build an affordable, reliable, and incentive-based system to foster the connections within and in between communities that do not have full access to the existing carsharing and ride-hailing systems.

5.1.1 Application in Underserved Communities and Justification

Transportation is a scarce resource in metropolitan Detroit: 40% of residents do not own cars, and 40% among them do not have access to vehicles (Firth, 2016). Despite the fact that surrounding suburban areas have more than five times the number of employment opportunities as the city of Detroit, only 9% of these jobs are taken by residents of Detroit. In Detroit areas, more than 10,000 residents face tremendous pressure in commuting to their jobs in suburban communities that do not offer public transit. Even worse, some suburban municipalities decide to reduce and eliminate much of its public bus service due to financial and safety concerns (McLaughlin, 2015).

Residents in underserved communities are experiencing financial, technical, skill-based, informational, and social barriers to the use of shared mobility service. For both service providers and users, the adoption of the existing shared mobility forms in underserved

communities posts significant challenges. From service providers' perspective, income and crime rate have influence on the willingness of drivers to serve particular areas, and therefore residents there find fewer drivers and higher service price (Thebault-Spieker et al., 2015). Meanwhile, due to the lack of personal vehicles, residents in low-income communities are disadvantaged in finding work as drivers for ride-hailing services (Gross et al., 2012).

For validating the design of CRS, we consider a service region in underserved communities that mainly involves, e.g., underserved populations such as jobless, elderly, and disabled, to whom transportation is a scarce resource. We partition the service region into zones and shared cars are located *a priori* in some designated parking spaces in each zone. We classify two types of users: Type 1 who want shared cars for private use but also have spare time outside their travel time windows to serve as drivers, and Type 2 who have ride-hailing demand but cannot drive themselves. We aim to build a self-sustained CRS system to encourage Type 1 drivers to "serve" Type 2 users by enabling them to receive income for providing rides to compensate for their payment for car rental.

To implement CRS, we build a reservation system that collects Type 1 drivers' rental information along with their available time windows for serving others, as well as Type 2 users' ride-hailing requests with origin-destination of their trips and time windows for pick-up. Note that both carsharing and ride-hailing have two primary forms: reservation-based and on-demand. From the users' perspective, on-demand service is more appealing since it provides an instant solution for their travel needs. However, in this chapter, we assume that all the requests are collected prior to the optimization phase due to the following reasons. First, on-demand ride-hailing service usually requires a large number of drivers and passengers to achieve matching efficiency. We consider travel needs from underserved communities, and thus the demand is sparsely distributed. Second, Type 1 drivers who serve others to gain more vehicle access often know their available time sufficiently early before committing to the service. Third, the purposes of the users' trips can be generally

categorized as job commute, hospital visit, job interview, and grocery shopping, which all have fixed schedule and can be known in advance. Based on this, we propose a reservation-based CRS system and assume that all supplies and demands are known in advance.

5.1.2 Contributions and Main Results

This chapter focuses on developing a new CRS system for serving heterogeneous travelers whose demand cannot be solely met by either carsharing or ride-hailing business models. The contributions of this chapter are two-fold.

- The classification of two types of travelers leads to self-sustained operations of the system via supply-demand matching. However, combining two types of services results in substantial operational challenges. We decompose the problem into two phases, and successfully reduce operational complexity while obtaining good-quality results shown by our computational studies. We also demonstrate the importance of vehicle relocation, as we can achieve higher demand fulfillment rates even when we only allow round trips of shared cars.

- We extend the basic model to a stochastic integer program to capture the randomness of vehicle travel time and service time. We develop efficient decomposition algorithms for optimizing the large-scale stochastic model with finite samples. When applied to synthetic data, our proposed model achieves high operational efficiency (measured by waiting and overtime of the system), compared with the deterministic model using the expected values.

The proposed community-based CRS system can be applied beyond the scope of serving underserved populations as it provides a solution for communities where there is a mixture of travelers with different driving abilities and time flexibility. We note that the work in this chapter has been published in Yu and Shen (2020).

5.1.3 Structure of the Chapter

The remainder of this chapter is organized as follows. In Section 5.2, we review the literature related to carsharing, ride-hailing, and their optimization models and algorithms. In Section 5.3, we describe the problem and formulate a two-phase approach to optimize the integrated CRS design and operations. We further present a stochastic programming variant of the Phase II model by considering random vehicle travel time and service time. In Section 5.4, we apply an efficient way to decompose the proposed models and derive valid cuts based on the integer L-shaped method. In Section 5.5, we demonstrate the effectiveness of the CRS system and present computational results in various instances. We conclude the chapter and present future research directions in Section 5.6.

5.2 Literature Review

For carsharing, Laporte et al. (2015) classify the literature of carsharing under five main topics: station location, fleet dimensioning, station inventory, rebalancing incentives, and vehicle repositioning. Nourinejad and Roorda (2014) propose a dynamic optimization-simulation model to study the relationship between fleet size and reservation time (i.e., the time between reservation and picking up vehicles). Nair and Miller-Hooks (2014) use a bilevel mixed-integer linear program to optimize station location and capacity, as well as vehicle inventories. To handle vehicle imbalance in the one-way carsharing system, Kek et al. (2009) introduce a spatial-temporal network to model the movement of vehicles and determine the workforce needed for relocation. Similar approaches have been used by de Almeida Correia and Antunes (2012) to optimize parking locations, and by Fan (2014) to optimize the allocation and relocation in carsharing systems that allow one-way car rentals.

For ride-hailing, Agatz et al. (2012) conduct a comprehensive survey of optimization approaches for fleet management and other related operational problems in dynamic ride-hailing. Alonso-Mora et al. (2017) propose a general model for a dynamic real-time high-

capacity ride-pooling system that solves an assignment problem on a graph of feasible trips and compatible vehicles. To match ride-hailing requests to available cars, as well as to route shared vehicles, most papers consider variants of the VRP for seeking static ride-hailing solutions. Toth and Vigo (2014) summarize the formulations and solution approaches of VRP variants, including multi-depot VRP, VRP with time windows, and VRP with simultaneous pickup and delivery. Taş et al. (2013) develop heuristic approaches for VRP with soft time windows and stochastic travel time. In addition to routing cost, they also consider the cost of late and early arrivals, which are related to the waiting time and idle time considered in our stochastic formulation discussed later.

For designing carsharing systems, He et al. (2016) optimize service zone selection for sharing electric vehicles under uncertain carsharing demand and fuel price. Brandstatter et al. (2016) further study facility location problems for locating charging stations in electric vehicle sharing systems. Lu et al. (2018) study a carsharing fleet allocation problem with stochastic one-way and round-trip demands. They propose a two-stage stochastic program to minimize the total cost of parking lots/permits and car allocation, as well as penalty cost from unfulfilled demand. Zhang et al. (2018) extend the models and solution approaches in Lu et al. (2018) for optimizing fleet allocation and service operations of electric vehicle sharing with vehicle-to-grid selling under random travel demand and electricity price. To the best of our knowledge, this chapter is the first to combine carsharing system design with ride-hailing service optimization, which has special application domains as we have described above.

5.3 Problem Formulations

We consider a fleet of K shared vehicles available to be reserved that are distributed in I zones. Let $L = \{1, 2, \ldots, |L|\}$ be a set of reservations received from Type 1 drivers. Each $l \in L$ is associated with a tuple, $(o_l,\ d_l,\ [s_l,\ t_l],\ [g_l, h_l])$, which includes the pick-up

zone $o_l \in I$, return zone $d_l \in I$ of a rental car, time window $[s_l, t_l]$ during which a car is needed, and the time window $[g_l, h_l]$ during which Type 1 driver l is able to provide ride-hailing service to Type 2 users. Let J be a set of ride-hailing demand from Type 2 (non-driver) users. Each $j \in J$ is associated with a tuple, $(o'_j,\ d'_j,\ e_j,\ g'_j,\ h'_j)$, where o'_j and d'_j represent the trip's origin and destination, respectively, e_j is the total service time needed (including driving time from o'_j to d'_j and the time of loading passengers), and $[g'_j, h'_j]$ is the available time window for picking up the corresponding Type 2 user at origin o'_j. In this chapter, we consider finite and discrete time for decision making in our models. We define a binary parameter vector $w = (w_{jl}, j \in J,\ l \in L)^T$, where $w_{jl} = 1$ if ride-hailing request $j \in J$ can be served by carsharing trip $l \in L$ and 0 otherwise. We set $w_{jl} = 1$ if $|[g'_j, h'_j + e_j] \cap [g_l, h_l]| \geq e_j$, meaning that Type 2 user $j \in J$ can be served within the time window specified by Type 1 driver $l \in L$.

For each Type 1 demand $l \in L$, we charge r_l^{car} per period of car use, dependent on pick-up and drop-off locations. Each driver l also earns r_l^{drive} for every time period of serving a Type 2 user. For each Type 2 demand $j \in J$, we charge r_j^{ride} per period dependent on the origin, destination, and time window. A car in use will incur a service cost c^{ser} (including, estimated gas, maintenance, insurance, and other types of cost) per period and will incur an idle cost c_i^{idle} (including c^{ser} and parking cost) per period if it sits idle at zone $i \in I$. When a car needs to be relocated, it will incur a relocation cost c^{rel} per hour. Note that service cost c^{ser}, idle cost c_i^{idle}, and relocation cost c^{rel} are not paid by Type 1 driver, but are system-based costs to minimize.

5.3.1 Solution Approach Overview

We consider a two-phase approach for the design and operations of the CRS system: in Phase I, we maximize the fulfillment of Type 1 demand over a spatial-temporal network that describes all users' demand and availability, while maintaining the financial self-sustainability of the system. We prioritize Type 1 drivers with a high possibility of pro-

viding rail-hailing service with a preliminary match of Type 1 drivers and Type 2 users when making the decision. In Phase II, we match Type 1 and Type 2 users and optimize pick-up routes and schedules under stochastic vehicle travel time and passenger loading (service) time, which may result in users' waiting and overtime. By decomposing the problem into two phases, it reduces the size of the ride-hailing scheduling and routing problem by only considering the accepted Type 1 drivers and hence improves the computational time overall.

We also develop a two-stage stochastic mixed-integer programming formulation for Phase II, which balances vehicles' total travel cost and the expected penalty cost of users' waiting and vehicle-use overtime. To solve the large-scale formulation with many samples of uncertain travel time and service time, we propose a driver-based decomposition algorithm, which first determines a sequence of Type 2 users assigned to each Type 1 driver, and then, for each driver, decides an optimal schedule to arrive at each Type 2 user's location, which can be quickly solved via linear programming. We conduct computational studies by testing instances based on serving underserved populations in Washtenaw County, Michigan, and consider diverse demand patterns of Type 1 and Type 2 users. We show that our decomposition algorithm outperforms the standard Benders decomposition approach, and demonstrate the cost and service-quality performance of the CRS system.

5.3.2 Phase I: Carsharing Planning and Operations

Given service requests from Type 1 drivers and Type 2 users, we first implement Phase I to decide which Type 1 requests to fulfill and pass the solutions to Phase II. We construct a spatial-temporal network, $G = (N, A)$, to capture carsharing reservations from Type 1 drivers over T periods, where each node $n_{it} \in N$ represents a zone $i \in I$ at period $t \in \{0, 1, \ldots, T\}$. We partition A into three types of arcs, $A = A^{\text{travel}} \cup A^{\text{idle}} \cup A^{\text{rel}}$, as follows.

- Travel arcs $(n_{it},\ n_{i't'}) \in A^{\text{travel}}$ are created for each Type 1 demand $l \in L$ where $i = o_l$, $i' = d_l$, $t = s_l$ and $t' = t_l$. The amount of flow on arc $a_l \in A^{\text{travel}}$ indicates

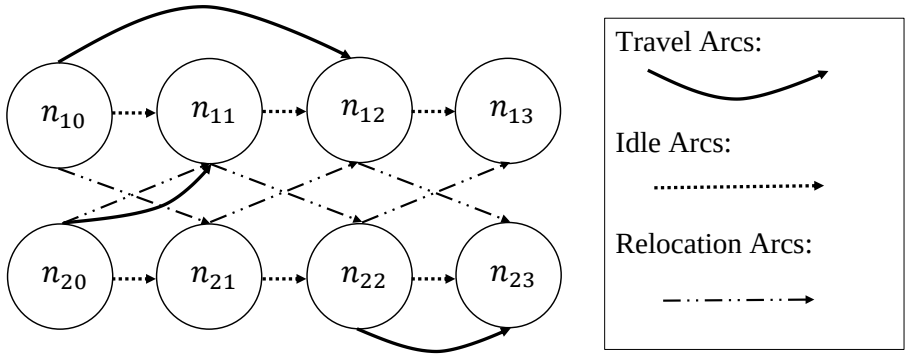


Figure 5.1: An example spatial-temporal network with two zones and three periods

whether or not a vehicle is being rented by Type 1 driver $l \in L$. We set $f_{a_l} = (t_l - s_l)(r_l^{\text{car}} - c^{\text{ser}}) - r_l^{\text{drive}}(h_l - g_l)$ as the unit flow revenue and $u_{a_l} = 1$ as the capacity of arc $a_l \in A^{\text{travel}}$, for all $l \in L$.

- Idle arcs $(n_{it},\ n_{i,t+1}) \in A^{\text{idle}}$ are created for $i \in I$ and $t \in \{0,\ 1, \ldots, T-1\}$. The amount of flow on arc $(n_{it},\ n_{i,t+1}) \in A^{\text{idle}}$ indicates the number of vehicles being idle in zone i from t to $t+1$. We set unit flow revenue and capacity of arc $a \in A^{\text{idle}}$ as $f_a = -c_i^{\text{idle}}$ and $u_a = K$, respectively.

- Relocation arcs $(n_{it}, n_{i',t'}) \in A^{\text{rel}}$ are created for $i \neq i' \in I$ and $t' > t \in \{0,\ 1, \ldots, T-1\}$. The amount of flow on $(n_{it}, n_{i',t'}) \in A^{\text{rel}}$ indicates the number of vehicles being relocated from zone i at t to zone i' at t'. We set unit flow revenue and capacity of arc $a \in A^{\text{rel}}$ as $f_a = -c^{\text{rel}}(t' - t)$ and $u_a = K$, respectively.

Figure 5.1 shows a spatial-temporal network example with $I = \{1, 2\}$ and $T = 3$. In the network, there are three Type 1 driver demands: a round trip reserving a car from $t = 0$ to $t = 2$ at zone 1, a round trip demand from $t = 2$ to $t = 3$ at zone 2, and a one-way demand from $t = 0$ to $t = 1$ traveling from zone 2 to zone 1. Note that by allowing vehicle relocation, we can use two vehicles to serve the three demand trips in the example network.

We define integer decision vector $x = (x_i,\ i \in I)^T$ where x_i is the number of cars initially located in zone $i \in I$, integer decision vector $y = (y_a,\ a \in A)^T$ where y_a indicates

the amount of flow on arc $a \in A$, and binary integer decision vector $z = (z_{jl}, j \in J,\ l \in L)^T$ where $z_{jl} = 1$ indicates that Type 2 user $j \in J$ can be potentially served by Type 1 driver $l \in L$, and $z_{jl} = 0$ otherwise. Let $\delta^+(n_{it})$, $\delta^-(n_{it})$ be the sets of arcs to which node n_{it} is their tail and head node, respectively, and formulate an integer program P1 as follows.

$$\textbf{[P1]} \quad \text{maximize}_{x,y,z} \quad \sum_{a\in A} f_a y_a + \sum_{j\in J} r_j^{\text{ride}} \sum_{l\in L} e_j z_{jl} \tag{5.1a}$$

$$\text{subject to} \quad \sum_{i\in I} x_i \le K \tag{5.1b}$$

$$\sum_{a\in\delta^+(n_{it})} y_a - \sum_{a\in\delta^-(n_{it})} y_a = \begin{cases} x_i & \text{if } t = 0 \\ 0 & \text{if } t \in \{1,\dots,T-1\} \\ -x_i & \text{if } t = T \end{cases}, \ \forall i \in I \tag{5.1c}$$

$$y_a \le u_a \qquad \forall a \in A \tag{5.1d}$$

$$\sum_{j\in J} z_{jl} e_j \le y_{a_l}(h_l - g_l) \qquad \forall l \in L \tag{5.1e}$$

$$z_{jl} \le w_{jl} \qquad \forall j \in J,\ \forall l \in L \tag{5.1f}$$

$$\sum_{l\in L} z_{jl} \le 1 \qquad \forall j \in J \tag{5.1g}$$

$$x \in \mathbb{Z}_{\ge 0}^{|I|},\ y \in \mathbb{Z}_{\ge 0}^{|A|},\ z \in \{0,1\}^{|J|\times|L|}. \tag{5.1h}$$

The objective function (5.1a) maximizes the potential total profit from both carsharing and ride-hailing operations, which is also equivalent to maximizing the total number of Type 1 and Type 2 requests served in the system. Constraint (5.1b) allows the total number of vehicles used in the CRS system to be no more than K. Constraints (5.1c) are flow balance constraints formulated for the spatial-temporal network. Constraints (5.1d) are arc capacity constraints. Constraints (5.1e) ensure that the accepted carsharing requests from all Type 1 drivers can potentially provide sufficient time to serve accepted ride-hailing requests from all Type 2 users. Constraints (5.1f) ensure that $z_{jl} = 1$ only if a Type 1 driver $l \in L$ can serve a Type 2 user $j \in J$. Constraints (5.1g) ensure that each ride-hailing request will be served at most once.

5.3.3 Phase II: Ride-hailing Routing and Scheduling

After solving P1, we obtain a set of Type 1 drivers to accept (i.e., all the $l \in L$ with $y_{a_l} = 1$ in the optimal solution). We also locate vehicles in zones following the values of x-variables in the solution but discard the values of z-variables. We then construct a Phase II model to find the routing and scheduling decisions for both Type 1 and Type 2 requests that can be accepted. Note that our final service acceptance and driver-user matching decisions will be made in Phase II, and could be different from the estimated solutions in Phase I. In practice, both types of users will be notified whether we can provide carsharing or ride-hailing service to them after we solve Phase II.

We define a network based on the CRS service region as $G' = (V, E)$ where V is the node set and $E = \{(u, v) : u, v \in V,\ u \neq v\}$ is the edge set. Let L' be the set of accepted Type 1 reservations (given by the subset of drivers in L with $y_{a_l} = 1$ in the optimal solution of Phase I), and $V = V_0 \cup V_1 \cup V_2$ where $V_0 = \{o_l :\ l \in L'\}$ and $V_1 = \{d_l :\ l \in L'\}$ are the sets of pick-up and return locations for approved Type 1 carsharing requests from solving P1, respectively. Set $V_2 = \{v_j :\ j \in J\}$ contains each Type 2 ride-hailing request. We assume that all the arcs can be traveled along both directions but can have different travel time, and therefore G' is directed. We use c_{uv} to denote the estimated travel time between two nodes such that $(u, v) \in E$. For each $v_j \in V_2,\ j \in J$, we calculate the estimated travel time in the following way: we set $c_{u,v_j} = c_{u,o'_j}$ and $c_{v_j,u} = c_{d'_j,u}$ for all $u \in V$, where $o'_j,\ d'_j$ represent origin and destination of Type 2 user $j \in J$.

We define binary decision vector $\alpha = (\alpha^l_{uv},\ (u, v) \in E,\ l \in L')^T$ such that $\alpha^l_{uv} = 1$ indicates that Type 1 driver $l \in L'$ travels along arc (u, v), and 0 otherwise. We define binary vector $\beta = (\beta_j,\ j \in J)^T$ such that $\beta_j = 1$ indicates that Type 2 user $j \in J$ is served, and 0 otherwise. We define continuous decision vector $\gamma = (\gamma_v,\ v \in V)^T$ such that $\gamma_v \geq 0$ is the planned time of a vehicle arrival at location $v \in V$. We formulate the Phase

II problem as follows, using a variant of the VRP with time windows and multiple depots.

$$\text{[P2]} \quad \text{maximize}_{\alpha,\beta,\gamma} \quad \sum_{j\in J} r_j^{\text{ride}} e_j \beta_j \tag{5.2a}$$

$$\text{subject to} \quad \sum_{l\in L'}\sum_{u\in V} \alpha^l_{uv_j} = \beta_j \qquad \forall v_j \in V_2, \tag{5.2b}$$

$$\sum_{v:(o_l,v)\in E} \alpha^l_{o_l v} - \sum_{v:(v,o_l)\in E} \alpha^l_{v o_l} = 1 \qquad \forall o_l \in V_0,\ l \in L' \tag{5.2c}$$

$$\sum_{u:(u,d_l)\in E} \alpha^l_{u d_l} - \sum_{u:(d_l,u)\in E} \alpha^l_{d_l u} = 1 \qquad \forall d_l \in V_1,\ l \in L' \tag{5.2d}$$

$$\sum_{u:(u,v)\in E} \alpha^l_{uv} - \sum_{u:(v,u)\in E} \alpha^l_{vu} = 0 \qquad \forall v \in V_2,\ l \in L' \tag{5.2e}$$

$$\alpha^l_{uv} = 0 \qquad \forall u \in V_0,\ v \in V,\ u \neq o_l,\ l \in L' \tag{5.2f}$$

$$\gamma_{o_l} + c_{o_l v} - T\left(1 - \alpha^l_{o_l v}\right) \leq \gamma_v \qquad \forall o_l \in V_0,\ (o_l, v) \in E, \tag{5.2g}$$

$$\gamma_{v_j} + e_j + c_{v_j u} - T\left(1 - \sum_{l\in L'} \alpha^l_{v_j u}\right) \leq \gamma_u \qquad \forall v_j \in V_2,\ (v_j, u) \in E, \tag{5.2h}$$

$$g_l \leq \gamma_{o_l} \leq \gamma_{d_l} \leq h_l \qquad \forall l \in L', \tag{5.2i}$$

$$g'_j \leq \gamma_{v_j} \leq h'_j \qquad \forall v_j \in V_2, \tag{5.2j}$$

$$\alpha \in \{0,1\}^{|E|\times|L'|}, \beta \in \{0,1\}^{|J|}, \tag{5.2k}$$

where the objective function (5.2a) maximizes the total profit from providing ride-hailing services. Constraints (5.2b) ensure that $\beta_j = 1$ if the origin o'_j for Type 2 user $j \in J$ has been visited. Constraints (5.2c)–(5.2e) balance vehicle flow for each approved Type 1 driver $l \in L'$. Constraints (5.2f) forbid flow for approved Type 1 driver $l \in L'$ to visit nodes representing pickup/return location for Type 1 driver $l' \in L'$, for all $l' \neq l$. Constraints (5.2g)–(5.2h) formulate the arrival time at each origin v of Type 2 user to be greater than the arrival time at node u (which is either pickup location of a Type 1 driver or destination location of another Type 2 user) plus travel and service time, if a trip travels from u to v directly. Constraints (5.2i)–(5.2j) ensure that departure and return time for Type 1 driver,

as well as departure time for each Type 2 user fall into the corresponding time windows.

5.3.4 Phase II Model Variant under Stochastic Travel Time and Service Time

In practice, uncertainties exist and affect the operations of the proposed CRS system. For example, travel time and service time can be random due to varying road conditions, weather, and traffic. Therefore, the primary task in this section is to propose a two-stage stochastic programming formulation that incorporates random travel and service time for Phase II.

Let $\tilde{c}_{uv}$ be the random travel time along arc $(u, v) \in E$ and $\tilde{e}_j$ be the random service time for ride $j \in J$. Under uncertainty, one or both of the following scenarios could happen: (i) Type 1 driver may arrive late to pick up the scheduled Type 2 user and (ii) Type 1 driver may return the shared vehicle later than the scheduled returning time. Therefore, our goal is to optimize the start time of each ride to maximize the revenue from ride-hailing operation minus the expected penalty cost due to Type 2 users' waiting and system overtime. (We define the overtime as the sum of the late time of returning vehicles by all Type 1 drivers.) We denote p^{w} and p^{o} as the unit penalty cost of waiting and overtime, respectively.

Let $\bar{e}_j$ be an estimated service time for Type 2 user $j \in J$, which can be taken as the mean value of $\tilde{e}_j$. We revise P2 and present its stochastic variant as:

$$\textbf{[SP2]} \quad \underset{\alpha,\beta,\gamma}{\text{maximize}} \sum_{j \in J} r_j^{\text{ride}} \bar{e}_j \beta_j - \mathbb{E}\left(Q(\alpha, \gamma, \tilde{c}, \tilde{e})\right) \tag{5.3}$$

$$\text{subject to: (5.2b)–(5.2k)}$$

where $Q(\alpha, \gamma, \tilde{c}, \tilde{e})$ is the total penalty cost of random waiting time and overtime given solution (α, γ) and uncertainty $(\tilde{c}, \tilde{e})$, and $\mathbb{E}(\cdot)$ denotes the expectation of random variable $\cdot$. To approximate $\mathbb{E}(Q(\alpha, \gamma, \tilde{c}, \tilde{e}))$, we apply the Sample Average Approximation (SAA)

method (Kleywegt et al., 2002). The idea is to generate a finite set of samples following the Monte Carlo sampling approach and approximate the expectation of its sample average function.

Let Ω denote the set of all sampled scenarios, with the probability of realizing each scenario is $1/|\Omega|$ when applying the SAA approach, i.e.,

$$\mathbb{E}\left(Q(\alpha,\gamma,\tilde{c},\tilde{e})\right) = \sum_{\omega\in\Omega}\frac{1}{|\Omega|}Q(\alpha,\gamma,\tilde{c}(\omega),\tilde{e}(\omega)),$$

where $\tilde{c}(\omega),\ \tilde{e}(\omega)$ are realizations of $\tilde{c},\ \tilde{e}$ in scenario ω, respectively. We define auxiliary decision variables $W_j(\omega)$ as waiting time for Type 2 ride request $j \in J$, and $O_l(\omega)$ as overtime for Type 1 driver service $l \in L'$ for each scenario $\omega \in \Omega$. Given an (α,γ)-solution and realized value $(\tilde{c}(\omega),\tilde{e}(\omega))$ of $(\tilde{c},\tilde{e})$ in scenario $\omega \in \Omega$, we specify the sample-based linear program for computing the value of $Q(\alpha,\gamma,\tilde{c}(\omega),\tilde{e}(\omega))$ as

$$\text{minimize } \sum_{j\in J} p^{\text{w}} W_j(\omega) + \sum_{l\in L'} p^{\text{o}} O_l(\omega) \tag{5.4a}$$

$$\text{subject to } \gamma_u + \tilde{c}(\omega)_{uv} - T\left(1-\alpha^l_{uv}\right) \le \gamma_v + W_j(\omega) \qquad \forall (u,v) = (o_l, o'_j) \in E, \tag{5.4b}$$

$$\gamma_{o'_j} + W_j(\omega) + \tilde{e}_j(\omega) + \tilde{c}_{d'_j o'_{j'}}(\omega) - T\left(1 - \sum_{l\in L'} \alpha^l_{d'_j o'_{j'}}\right) \le \gamma_{o'_{j'}} + W_{j'}(\omega) \qquad \forall (d'_j, o'_{j'}) \in E, \tag{5.4c}$$

$$\gamma_{o'_j} + W_j(\omega) + \tilde{e}_j(\omega) + \tilde{c}_{d'_j d_l}(\omega) - T\left(1-\alpha^l_{d'_j d_l}\right) \le \gamma_{d_l} + O_l(\omega) \qquad \forall (d'_j, d_l) \in E, \tag{5.4d}$$

$$W_j(\omega) \ge 0 \qquad \forall j \in J, \tag{5.4e}$$

$$O_l(\omega) \ge 0 \qquad \forall l \in L'. \tag{5.4f}$$

Here the objective function (5.4a) minimizes the total penalty cost of all Type 2 users' waiting time and all Type 1 drivers' overtime of returning vehicles. Let $S_j(\omega)$ be the actual service starting time for Type 2 user $j \in J$ and $S'_l(\omega)$ be the actual vehicle return time for Type 1 driver $l \in L'$ for each sampled scenario $\omega \in \Omega$, respectively. We have

- $S_j(\omega) = \gamma_v + W_j(\omega)$ for $v \in V_2$ representing Type 2 user $j \in J$, and sampled scenario $\omega \in \Omega$;

- $S'_l(\omega) = \gamma_v + O_l(\omega)$ for $v \in V_1$ representing Type 1 driver $l \in L'$, and sampled scenario $\omega \in \Omega$.

Note that $S_j(\omega)$ and $S'_l(\omega)$ are related to the right-hand sides of constraints (5.4b)–(5.4c) and (5.4d), respectively. Therefore, for each Type 1 driver $l \in L'$, the corresponding constraint (5.4b) calculates the actual service start time of its first served Type 2 user j and ensures that it is no earlier than the planned time that driver l departs origin o_l plus the travel time from o_l to o'_j. Similarly, each constraint in (5.4c) propagates this time relationship for all the subsequent Type 2 users who will be served by Type 1 driver l, and ensures that their actual service start time will be no earlier than the actual service start time of their predecessors plus the realized service time and travel time before driver l arrives. Lastly, constraint (5.4d) calculates the actual time of returning the vehicle by Type 1 driver $l \in L'$ and ensures that it is no earlier than the time of completing service in the last Type 2 user's location plus the travel time to the return location.

5.4 Solution Approaches

5.4.1 An Integer L-shaped Approach

To solve SP2 efficiently, we propose an algorithm based on the integer L-shaped method (Laporte and Louveaux, 1993). We decompose the problem into a relaxed master problem, which contains variables of all the decisions made before realizing the values of $(\tilde{c}, \tilde{e})$, and a series of subproblems with recourse decisions. Unlike the standard decomposition approach that creates a subproblem for each scenario, we will reformulate our problem by first deciding routes for individual drivers and then creating driver-based subproblems. Specifically, the master problem matches and assigns sequences of Type 2 users to each Type 1 driver. Subproblems are then formulated for each Type 1 driver, to find an optimal schedule to pick up assigned Type 2 users, and each subproblem aims to minimize the expected penalty cost of the assigned Type 2 users' total waiting time and the corresponding

Type 1 driver's overtime use of the vehicle. We initially set the lower bound of the expected penalty cost for each Type 1 driver as zero. Then we iteratively generate cuts from each subproblem and add them to the master problem to improve the lower bound.

We define decision variables $\theta = (\theta_l, l \in L')^T$, such that θ_l is the optimal objective value (i.e., the optimal expected penalty cost of waiting time and overtime) of the subproblem formulated for Type 1 driver l, for each $l \in L'$. We formulate a relaxed master problem in the current iteration as:

$$\textbf{[MP]} \quad \underset{\alpha,\beta,\theta}{\text{maximize}} \sum_{j \in J} r_j^{\text{ride}} \bar{e}_j \beta_j - \sum_{l \in L'} \theta_l \tag{5.5a}$$

$$\text{subject to: (5.2b)–(5.2k)}$$

$$L(\alpha, \theta_l) \geq 0 \qquad \forall l \in L' \tag{5.5b}$$

where in (5.5b), $L(\alpha, \theta_l) \geq 0$ denotes the set of cuts for improving the lower bound of θ_l, generated from solving subproblems (described later) in previous iterations. In the above model, constraints (5.2b)–(5.2f) enforce sufficient vehicles to cover matched Type 2 users as explained in the previous section.

After solving **MP**, we obtain a tentative optimal solution $\bar{\alpha}$ that can be used to recover the routing sequence for each driver $l \in L'$ as follows. We use the non-zero values of $\bar{\alpha}_{uv}^l$ for $v \in V_2$ to obtain the set of Type 2 users assigned to driver $l \in L'$ and then perform a depth-first search to recover the path to visit assigned Type 2 users for each Type 1 driver. For each $l \in L'$, let $(\sigma^l(1), \sigma^l(2), \ldots, \sigma^l(n_l))$ be such a sequence with n_l denoting the total number of Type 2 users assigned to driver l. Let $\sigma^l(0)$ and $\sigma^l(n_l + 1)$ be the depots where Type 1 driver $l \in L'$ picked up and returned the vehicle. The only decisions left are the planned arrival time at each Type 2 user's pick-up location and the planned time for returning each vehicle. Alternatively, for each Type 1 driver $l \in L'$, we can decide the time to allocate in between $\sigma^l(n)$ and $\sigma^l(n + 1)$ for $n = 0, \ldots, n_l$.

Recall that $S_i(\omega)$ is the actual service start time at node i if $i \in V_2$ or actual vehicle

return time at node i if $i \in V_1$. The subproblem for each Type 1 driver $l \in L'$ is equivalent to the following linear program:

$$\textbf{[SUBP}_l\textbf{]} \quad \underset{W,O,S,\gamma}{\text{minimize}} \ \frac{1}{|\Omega|}\left(\sum_{i=1}^{n_l} p^{\text{w}} W_{\sigma^l(i)}(\omega) + p^{\text{o}} O_l(\omega)\right) \tag{5.6a}$$

subject to: (5.2i), (5.2j)

$$S_{\sigma^l(1)}(\omega) \geq \gamma_{\sigma^l(0)} + c_{\sigma^l(0),\sigma^l(1)}(\omega) \qquad \omega \in \Omega, \tag{5.6b}$$

$$S_{\sigma^l(i+1)}(\omega) \geq S_{\sigma^l(i)}(\omega) + e_{\sigma^l(i)}(\omega) + c_{\sigma^l(i),\sigma^l(i+1)}(\omega) \qquad \omega \in \Omega,\ i = 1,\ldots,n_l, \tag{5.6c}$$

$$S_{\sigma^l(i)}(\omega) = \gamma_{\sigma^l(i)} + W_{\sigma^l(i)}(\omega) \qquad i = 1,\ldots,n_l,\ \omega \in \Omega, \tag{5.6d}$$

$$S_{\sigma^l(n_l+1)}(\omega) = \gamma_{\sigma^l(n_l+1)} + O_l(\omega) \qquad \omega \in \Omega, \tag{5.6e}$$

$$W_{\sigma^l(i)}(\omega) \geq 0 \qquad \forall i = 1,\ldots,n_l,\ \omega \in \Omega \tag{5.6f}$$

$$O_l(\omega) \geq 0 \qquad \forall \omega \in \Omega, \tag{5.6g}$$

where constraints (5.6b) ensure that the actual service start time at the first assigned Type 2 user is not earlier than the actual time Type 1 driver leaves depot, plus the travel time from depot to the Type 2 user's pick-up location. Constraints (5.6c) ensure that the actual service start time at the $(i+1)^{\text{th}}$ node (or actual vehicle return time if such a node represents a depot) of Type 1 driver l is no earlier than the actual service start time at the i^{th} Type 2 user plus the time for completing the service at the i^{th} Type 2 user and the travel time from the i^{th} user to the $(i+1)^{\text{th}}$ node. For all scenarios $\omega \in \Omega$, constraints (5.6d) let the actual time of serving each assigned Type 2 user be the planned arrival time plus the waiting time of the user. Constraints (5.6e) let the actual time for returning the vehicle be the planned time plus the overtime. Both waiting time and overtime variables are nonnegative according to (5.6f) and (5.6g). These constraints are analogous to constraints (5.4b)–(5.4d). Each subproblem $\textbf{SUBP}_l$ will output an optimal schedule for Type 1 driver $l \in L$ to serve the assigned Type 2 users (given by **MP**), and also the minimum expected penalty cost given the current visiting sequence. Moreover, $\textbf{SUBP}_l$ are linear programs without big-M coefficients, which can be solved very efficiently to return cuts to **MP**.

After solving **MP**, we obtain an integer solution $\bar{\alpha}$ as well as solutions $\bar{\theta}_l$ for all $l \in L$.

Let $\hat{\theta}_l$ be the optimal objective obtained by solving **SUBP**$_l$ for each $l \in L$. If $\bar{\theta}_l < \hat{\theta}_l$, following the integer L-shaped method (see, e.g., Laporte et al., 2002), we propose to add the following cut to the **MP** and re-solve it:

$$\theta_l \geq \hat{\theta}_l \left(\sum_{u,v:\bar{\alpha}^l_{uv}=1} \alpha^l_{uv} - \sum_{u,v} \alpha^l_{uv} + 1 \right). \tag{5.7}$$

Theorem 5.1. Cut (5.7) is a valid inequality for **MP** and enforces that no same values of $\bar{\alpha}, \bar{\theta}_l$ can be obtained in future iterations.

Proof. Inequality (5.7) only takes effect when $\sum_{u,v:\bar{\alpha}_{uv}=1} \alpha^l_{uv} - \sum_{u,v} \alpha^l_{uv} \geq 0$, which is equivalent to $\alpha^l_{uv} = 1$ for all u, v with $\bar{\alpha}^l_{uv} = 1$. Each subproblem **SUBP**$_l$ outputs the optimal scheduling for Type 1 driver l and calculates the minimum expected penalty cost, θ_l. The variable representing recourse penalty cost for the current visiting sequence should be bounded below by $\hat{\theta}_l$. Therefore, (5.7) is valid for any optimal (α, θ_l), and the same value of $(\bar{\alpha}, \bar{\theta}_l)$ will not repeat if it is not optimal. □

We summarize the algorithmic steps of the decomposition-based cutting-plane algorithm in Algorithm 5.1.

Theorem 5.2. Algorithm 5.1 converges in finitely-many steps.

Proof. In each iteration, we have $\bar{\theta}_l < \hat{\theta}_l$ only if current visiting sequence for driver $l \in L'$ has not been explored; otherwise cut (5.7) enforces $\bar{\theta}_l \geq \hat{\theta}_l$. Since we have a finite number of visiting sequences for a given Type 2-to-Type 1 assignment, the algorithm converges in finitely-many steps. □

Remark 5.1. To solve SP2, alternatively, we can decompose the problem by scenario (instead of decomposing the model by driver l, as proposed in Algorithm 5.1) and construct a relaxed master problem that matches Type 1 drivers to Type 2 users and determine pick-up routes and schedules, and then formulate subproblems to compute the penalty cost of Type 2 users' waiting time and Type 1 drivers' overtime in each scenario. However, this

Algorithm 5.1: An integer L-shaped method for solving SP2.

Initialize **MP** as (5.5a)–(5.5b)and set $L(\alpha, \theta_l) \geq 0 = \emptyset$ for all $l \in L'$;
Solve **MP** by branch-and-bound to obtain a tentative solution $(\bar{\alpha}, \bar{\theta})$;
Recover route sequence $(\sigma^l(1), \sigma^l(2), \ldots, \sigma^l(n_l))$ for each $l \in L'$ from $\bar{\alpha}$;
Solve **SUBP**$_l$ for each route sequence of driver $l \in L'$, and let $\hat{\theta}_l$ be the optimal objective for **SUBP**$_l$;
if $\sum_{l \in L'} \bar{\theta}_l < \sum_{l \in L'} \hat{\theta}_l$ **then**
    for $l \in L'$ **do**
        Add Cut (5.7) to the cut set $L(\alpha, \theta_l) \geq 0$ in **MP**;
    end
    goto Step 2.;
end
else
    Store the solution $\bar{\gamma}$ from each **SUBP**$_l$ as planned service start time for each assigned Type 2 user to driver l;
    Return solution $(\bar{\alpha}, \bar{\gamma}, \bar{\theta})$ as an optimal solution to the overall SP2 problem;
end

traditional way of decomposing the problem cannot produce sufficiently good routes and schedules in the master problem under uncertain travel time and service time, which will only be realized in the subproblems. Furthermore, compared to the constraints in SP2 that determine feasible routes and schedules, the driver-based decomposition algorithm as Algorithm 5.1 avoids "big-M" constraints in both MP and SUBP$_l$, for all $l \in L'$. For these two reasons, our decomposition approach significantly improves the solution time of SP2, which we will demonstrate later in numerical studies.

We may further decompose SUBP$_l$ by scenario. Since SUBP$_l$ only contains continuous decision variables and is a linear program that can be quickly solved to obtain cut (5.7), we directly solve it without further decomposition in our numerical studies.

5.4.2 Benchmark Approaches

To benchmark our proposed algorithm for SP2, we compare its computational results against two other approaches for solving SP2. The first approach is to solve the deterministic mixed-integer linear programming equivalent of SP2 using a general commercial

optimization solver. We allow the solver to detect possible decomposable structures itself and use any algorithmic routines embedded in it to handle the special structure of the problem. However, we do not specify which algorithms to use when directly solving the deterministic equivalent formulation.

The second approach to apply the Benders decomposition approach (see Birge and Louveaux, 2011) for solving the sample-based reformulation of SP2, which starts with a relaxed master problem containing only first-stage decision variables (made before realizing the values of uncertainties) and iteratively solves subproblems with recourse variables to generate and add cuts to the relaxed master problem. The cuts are for approximating the value function of the second-stage recourse problem in terms of first-stage decisions.

We detail the Benders decomposition procedures as follows. Let θ be a decision variable approximating the value function of recourse cost. For SP2, we start with the relaxed master problem:

$$\textbf{[RMP-Benders]} \quad \underset{\alpha,\beta,\gamma,\theta}{\text{maximize}} \sum_{j\in J} r_j^{\text{ride}} \bar{e}_j \beta_j - \theta \tag{5.8a}$$

$$\text{subject to (5.2b)–(5.2k)}$$

$$B(\alpha,\beta,\gamma,\theta) \geq 0, \tag{5.8b}$$

where constraints (5.8b) contains a set of Benders cuts that will be generated in later iterations. Initially, we only have $\theta \geq 0$. Note that the **RMP-Benders** is a mixed-integer linear program.

Subproblems are linear programs computing the recourse cost for each sampled scenario, i.e., the penalty cost of waiting time and overtime given the routing and scheduling decisions from **RMP-Benders**. For each scenario $\omega \in \Omega$, the recourse cost can be computed by (5.4a)–(5.4f). Here, we consider subproblems in the dual form of (5.4a)–(5.4f) by defining dual variables $\pi(\omega), \mu(\omega), \nu(\omega)$ associated with constraints (5.4b)–(5.4d), respectively. We formulate subproblems in their dual form for each scenaraio ω and iteration v

as:

$$
\begin{aligned}
[\textbf{SUBP}_B^v(\omega)] \quad \underset{\pi,\mu,\nu}{\text{minimize}} \; & \sum_{(o_l,o'_j)\in E} (\gamma_{o_l} + \tilde{c}(\omega)_{o_l,o'_j} - T\left(1-\alpha^l_{o_l,o'_j}\right) - \gamma_{o'_j})\pi_{o_l,o'_j}(\omega) \\
& + \sum_{(d'_j,o'_{j'})\in E} (\gamma_{o'_j} + \tilde{e}_j(\omega) + \tilde{c}_{d'_j o'_{j'}}(\omega) - T\left(1-\sum_{l\in L'}\alpha^l_{d'_j o'_{j'}}\right) - \gamma_{o'_{j'}})\mu_{d'_j,o'_{j'}}(\omega) \\
& + \sum_{(d'_j,d_l)\in E} (\gamma_{o'_j} + \tilde{e}_j(\omega) + \tilde{c}_{d'_j d_l}(\omega) - T\left(1-\alpha^l_{d'_j d_l}\right) - \gamma_{d_l})\nu_{d'_j,d_l}(\omega) \qquad (5.9a) \\
\text{subject to} \; & \sum_{l\in L'} \pi_{o_l,o'_j}(\omega) + \sum_{j'\in J}(\mu_{d'_{j'},o'_j}(\omega) - \mu_{d'_j,o'_{j'}}(\omega)) - \sum_{l\in L'}\nu_{d'_j,d_l}(\omega) \le p^{\text{w}} \qquad \forall j\in J \qquad (5.9b) \\
& \sum_{j\in J}\nu_{d'_j,d_l}(\omega) \le p^{\text{o}} \qquad \forall l\in L' \qquad (5.9c) \\
& \pi(\omega),\mu(\omega),\nu(\omega) \ge 0, \qquad (5.9d)
\end{aligned}
$$

Our problem has complete recourse as every feasible solution from the **RMP-Benders** leads to a finite objective value of each subproblem. Therefore, we only consider optimality cuts when approximating the recourse costs. Following strong duality, the recourse cost $Q(\alpha,\gamma,\tilde{c}(\omega),\tilde{e}(\omega))$ equals to the optimal objective value of $\textbf{SUBP}_B^v(\omega)$ in each iteration v and for each scenario ω. Let $(\pi^v(\omega),\mu^v(\omega),\nu^v(\omega))^{\mathsf{T}}$ be an optimal solution to $\textbf{SUBP}_B(\omega)$ in the v^{th} iteration of the Benders approach. If we have not closed the optimality gap, we add the following Benders cut into **RMP-Benders**:

$$
\begin{aligned}
\theta \ge & \frac{1}{|\Omega|}\sum_{\omega\in\Omega}\Big(\sum_{(o_l,o'_j)\in E} (\gamma_{o_l} + \tilde{c}(\omega)_{o_l,o'_j} - T\left(1-\alpha^l_{o_l,o'_j}\right) - \gamma_{o'_j})\pi^v_{o_l,o'_j}(\omega) \\
& + \sum_{(d'_j,o'_{j'})\in E} (\gamma_{o'_j} + \tilde{e}_j(\omega) + \tilde{c}_{d'_j o'_{j'}}(\omega) - T\left(1-\sum_{l\in L'}\alpha^l_{d'_j o'_{j'}}\right) - \gamma_{o'_{j'}}(\omega))\mu^v_{d'_j,o'_{j'}}(\omega) \\
& + \sum_{(d'_j,d_l)\in E} (\gamma_{o'_j} + \tilde{e}_j(\omega) + \tilde{c}_{d'_j d_l}(\omega) - T\left(1-\alpha^l_{d'_j d_l}\right) - \gamma_{d_l})\nu^v_{d'_j,d_l}(\omega)\Big). \qquad (5.10)
\end{aligned}
$$

Algorithm 5.2 describes detailed algorithmic steps of the Benders approach.

Algorithm 5.2: The Benders decomposition method for solving SP2.

Set $v = 0$;
Initialize **RMP-Bender** with constraints (5.8b) contains only $\theta \geq 0$;
while *True* **do**
    $v = v + 1$;
    Solve **RMP-Bender** to obtain optimal solution $\alpha^v, \gamma^v, \theta^v$;
    Solve **SUBP**$_B^v(\omega)$ for each $\omega \in \Omega$ to obtain optimal solution $\pi^v(\omega), \mu^v(\omega), \nu^v(\omega)$;
    if $\theta = \theta^v$ *does not satisfy equation* (5.10) **then**
        Add Cut (5.10) to (5.8b) and update **RMP-Bender**;
    end
    else
        Terminate the procedure and output $\alpha^v, \gamma^v, \theta^v$;
    end
end

5.5 Numerical Results

We conduct numerical studies and demonstrate the performance of proposed models using randomly generated instances based on statistical and census data of underserved populations in Washtenaw County, Michigan.

5.5.1 Experiment Design

We generate test instances based on the most updated United States Census data for Washtenaw County in Michigan, collected in April 2010 (see United States Census Bureau, 2010). In the dataset, Washtenaw County is divided into 100 different census tracts, and each contains the information of population and location (i.e., longitude and latitude of the geographical center). We assume that each census tract is represented by its geographical center and construct a corresponding network with 100 nodes. The travel time between any of two nodes can be obtained by accessing Google Maps API[1]. In the original dataset, we can get the population for each census tract grouped by different ages. Due to safety concerns, we enforce being 21-65 years old as the age requirement for Type 1 drivers to match

[1] https://developers.google.com/maps/documentation/distance-matrix/

the age requirement of most carsharing service providers (e.g., Zipcar requires drivers to be at least 21 years old[2]). To simulate Type 2 users in underserved populations, we consider the populations with age over 50 and with disability.

- **Problem Size:** To pick the carsharing service zones in set I, we sort the 100 nodes based on the population of Type 1 drivers in each node and select the first $|I|$ nodes. We set $|I| = 5$, and assume that the operational hours of our system are from 7 am to 7 pm, and thus $T = 12$ hours.

- **Type 1 Driver Data:** We simulate the set of Type 1 driver reservations, L, as follows. Consider $|L| = 40$ or 60. For each Type 1 driver reservation $l \in L$, we simulate the pick-up and return locations, o_l and d_l, from the node set I based on the density of populations that are jobless or whose annual incomes are below a certain threshold. For the car reservation time s_l of each Type 1 driver $l \in L$, we sample s_l uniformly from 7 am to 4 pm with 1-hour time interval. The time of private use of a car is then uniformly sampled from 0–2 hours with 1-hour time interval. We simulate Type 1 drivers' different flexibility levels of serving Type 2 users and consider two types of service-hour distributions: for Case (i), we generate the service hours for Type 1 drivers from $\{1, 2, 3, 4\}$ with equal probability 0.25. For Case (ii), we sample the service hours for Type 1 drivers from $\{1, 2, 3, 4\}$ with probabilities 0.1, 0.2, 0.3, and 0.4, respectively. For the rest of the chapter, we refer to Case (i) as the case with "regular drivers" and to Case (ii) as the case with "flexible drivers". Except for the tests in Section 5.5.2.2 and Section 5.5.2.4, we report the results for the case with "flexible drivers".

- **Type 2 User Data:** We consider instances with $|J| = 40, 60, 80$ for Type 2 ride-hailing demand. For each Type 2 user $j \in J$, we sample the origin node, o'_j, based on the population density of target Type 2 users (age over 50 years old and disabled) and

[2] `https://support.zipcar.com/`

sample the destination node, d'_j, uniformly over the 100 census tracts. At the same time, we avoid $o'_j = d'_j$ in all our sampled instances by re-sampling the destination node if it happens. We uniformly generate g'_j for each Type 2 user over the entire operational hours span (i.e., 7 am to 7 pm) and assign a 30-minute time window for each pick-up, i.e., $h'_j = g'_j + 30,\ \forall j \in J$.

- **Stochastic Travel Time:** In all our test instances, for each Type 2 reservation $j \in J$, we set $e_j = c_{o'_j, d'_j}$ plus a constant loading/unloading time that is the same for all $j \in J$ and thus can be omitted in the model. Following the standard VRP literature such as Laporte et al. (1992); Polus (1979), we consider the Gamma distribution for sampling the random travel time between pairs of locations. We assume that $\tilde{c}_{ij} = c_{ij}(0.8 + \xi)$ where ξ follows a Gamma distribution with shape parameter α and scale parameter λ, and c_{ij} is the deterministic travel time obtained from Google Maps API for arc $(i, j) \in E$. Then, following the definition of Gamma distribution, we have

$$E(\tilde{c}_{ij}) = (0.8 + \alpha\lambda)c_{ij},$$
$$Var(\tilde{c}_{ij}) = \alpha\lambda^2 c_{ij}.$$

  We choose parameters $\alpha = 0.2$ and $\lambda = 1$ to generate all the scenarios.

- **Other Parameters:** We set $r^{\text{car}} = \$8$ per hour and $r^{\text{ride}} = \$20$ per hour to match the price of current carsharing and ride-hailing services, e.g., Zipcar and Uber, in Washtenaw County. Note that the pricing strategy for a ride-hailing company is composed of two parts, mileage ($0.95/mile) and service time ($0.15/minute) in Washtenaw County[3]. Here we combine them together to obtain reasonable price settings. We set $r^{\text{drive}} = \$16$ per hour to reasonably incentivize Type 1 drivers to provide ride-hailing services. Also, $c^{\text{ser}} = \$1$ per hour and $c^{\text{idle}} = \$1$ per hour. The penalty cost parameters are set as $p^{\text{w}} = \$0.1$ per minute per user and $p^{\text{o}} = \$0.1$ per minute per car for

[3] https://www.uber.com/fare-estimate/

model SP2.

Basic Settings			
$\|I\|$	5	$\|L\|$	{40,60}
T	12	$\|J\|$	{40,60,80}
$\|\Omega\|$	500		
Stochastic Travel Time			
c_{ij}	average travel time obtained from Google Map API		
ξ	Follow Gamma Distribution with parameters $\alpha = 0.2$, $\lambda = 1$		
$\tilde{c}_{ij}$	$c_{ij}(0.8 + \xi)$		
Other Parameters			
r^{car}	\$8/hour	c^{idle}	\$1/hour
r^{ride}	\$20/hour	p^{w}	\$0.1/hour
r^{drive}	\$16/hour	p^{o}	\$0.1/hour
c^{ser}	\$1/hour		

Table 5.1: Summary of key parameters

Table 5.1 summarizes the important parameters used in the numerical studies. For each test instance described above, we generate five replications and report the average statistics unless otherwise noted.

All instances are programmed using Java 10. We call the solver Gurobi 8.0 to optimize all mixed-integer linear programming models. All programs are run on a desktop computer with Microsoft Windows 10 64-bit operating system, an Intel Core i7-6700K Central Processing Unit (CPU) with 4.0 GHz, and 32.0 GB RAM.

5.5.2 Computational Results

For our numerical experiments, we analyze our proposed model from the perspectives of computational time, quality of service, revenue and cost, and out-of-sample tests.

5.5.2.1 Computational Time

We present the computational results of the proposed model for various test instances, with five replications for each parameter combination (with the same number of users but different input data, e.g., locations and pick-up time windows). For each instance, we report the maximum, minimum, and average CPU time. The average CPU time is calculated based on solved replications. We choose parameters as described in Section 5.5.1 and 500 scenarios for the stochastic programming model SP2, and set the CPU time limit as 30 minutes for computing each replication.

In Tables 5.2 and 5.3, we report the CPU time (in seconds) of models P1, P2, and SP2 for solving each test instance with different number of available vehicles (K), number of Type 1 reservations ($|L|$), and number of Type 2 reservations ($|J|$). We report optimality gap for SP2 when reaching the time limit. For SP2, we report results for (i) directly solving the MILP model (see Columns "SP2 (Direct)"), (ii) applying the Benders decomposition (see Columns "SP2 (Benders)"), and (iii) applying our integer L-shaped method using driver-based decomposition (see Columns "SP2 (L-shaped)").

Table 5.2: CPU time (in seconds) for models P1 and P2

			P1			P2		
K	$\|L\|$	$\|J\|$	max	min	avg	max	min	avg
20	40	40	0.02	0.01	0.01	2.18	1.56	1.91
20	40	60	0.02	0.01	0.01	59.02	5.78	23.05
20	40	80	0.02	0.02	0.02	176.65	18.05	76.65
30	60	40	0.02	0.01	0.02	18.58	5.36	9.65
30	60	60	0.02	0.02	0.02	31.34	12.23	20.51
30	60	80	0.03	0.02	0.03	483.93	33.91	256.95

According to Tables 5.2 and 5.3, model P1 is easy to solve whereas P2 and SP2 are relatively difficult to optimize for both deterministic and stochastic cases. Besides, SP2 is more challenging to solve compared to P2, as 4 out of 6 test instance sizes cannot be solved to optimality within the time limit when the travel time and service time are both random.

In Table 5.2, the CPU time for P2 increases as the number of Type 1 and Type 2 reser-

Table 5.3: CPU time (in seconds) or optimality gap for models SP2 with different methods

			SP2 (Direct)			SP2 (Benders)			SP2 (L-shaped)		
K	$\|L\|$	$\|J\|$	max	min	avg	max	min	avg	max	min	avg
20	40	40	38.37%*	0.57%*	10.30%*	2.73%*	1.22%*	1.84%*	0.20%*	113.03	317.51
20	40	60	N/A	N/A	N/A	3.26%*	2.25%*	2.77%*	1.43%*	0.59%*	0.98%*
20	40	80	N/A	N/A	N/A	5.21%*	3.92%*	4.53%*	5.03%*	2.65%*	3.44%*
30	60	40	N/A	1.33%*	37.87%*	1.59%*	1.07%*	1.28%*	0.14%*	120.42	247.97
30	60	60	N/A	N/A	N/A	2.38%*	1.51%*	1.92%*	0.68%*	0.22%*	0.45%*
30	60	80	N/A	N/A	N/A	–	–	–	6.81%*	1.53%*	4.00%*

*: optimality gap is reported; N/A: no feasible solutions found within the 30-minute time limit;
–: model is not solvable due to machine memory limit.

vations increases. For example, given $|L| = 40$, the average CPU time increases from 1.91 seconds to 76.65 seconds as $|J|$ grows; given $|J| = 60$, the average CPU time increases from 9.65 seconds to 256.95 seconds as $|L|$ grows from 40 to 80. We also demonstrate the effectiveness of the proposed solution approach, as both CPU time and optimality gap have been significantly improved from those given by general-purpose solvers.

Although some instances cannot be solved to optimality when testing SP2, the optimality gap can be small when reaching the CPU time limit, i.e., when we set the optimization gap tolerance to 5%, most of the cases can be solved within the time limit. Comparing to the other two solution approaches, our proposed algorithm works well and outputs solutions with the minimum optimality gap. Furthermore, due to the driver-based decomposition, we avoid the out-of-memory issue that appears in the scenario-based Benders decomposition when solving large instances. In later sections, we will see that the number of served trips output by SP2 is similar to that of P2 while achieving a much higher quality of service for scheduling. The difficulty of closing the optimality gap for SP2 is due to the schedule adjustment that minimizes the expected penalty cost. Therefore, although SP2 cannot be solved to optimality, we can still use the output solutions of matched Type 1 drivers and Type 2 users, and the sequence of serving Type 2 users to schedule the operations in Phase II. Based on the reasonable instance size of the designed system, our tests show the feasibility and practicability of the proposed models.

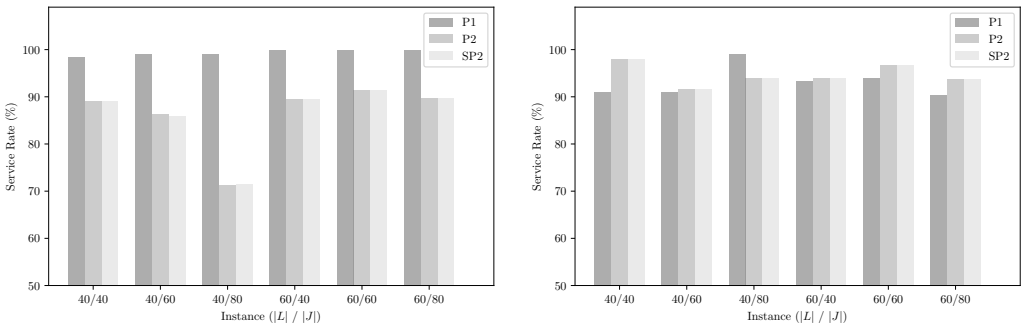
5.5.2.2 Quality of Service

Here we show the quality of service (QoS) results of the proposed models based on the percentage of approved demands. As mentioned in Section 5.5.1, we consider two cases of Type 1 drivers: "regular drivers" with service hours generated from $\{1, 2, 3, 4\}$ hours with equal probability and "flexible drivers" with service hours picked from $\{1, 2, 3, 4\}$ hours with probabilities $0.1, 0.2, 0.3, 0.4$, respectively, i.e., the latter case of drivers are more time flexible and are likely to provide service with extended hours. Recall that P1 yields the acceptance decisions for Type 1 driver and P2/SP2 yields the acceptance decisions for Type 2 users. We will measure the demand service rate for the corresponding user type for each model.

Figure 5.2a shows the average demand service rates across different models with "regular drivers" and Figure 5.2b shows the average demand service rates with "flexible drivers". For both cases, the demand service rate for Type 1 drivers is kept at high levels (over 90%). At the same time, both P2 and SP2 yield similar demand service rate for Type 2 users. However, for the case of "regular drivers", the demand service rate for Type 2 users varies from 70% to 90% due to the lack of available drivers/service hours: when the ratio between Type 2 users and Type 1 driver rises, the demand service rate for Type 2 users drops significantly (see the instance with $|L| = 40, |J| = 80$). On the other hand, the demand service rates for both Type 1 drivers and Type 2 users are very high (over 90%) across all instances for the case with "flexible drivers". Based on the results, the proposed model would be particularly helpful for the underserved communities whose residents are more flexible in terms of service hours.

5.5.2.3 Separated P1 and P2 versus Integrated Formulation

To demonstrate the benefits of using the two-phase approach that sequentially solves models in Phases I and II, we compare it with directly solving an integrated model that makes all decisions simultaneously (including vehicle allocation, Type 1 and Type 2 users to ac-



(a) Case with "regular drivers"

(b) Case with "flexible drivers"

Figure 5.2: Average demand service rates by proposed models

cept, and the corresponding matching and routing decisions). In particular, we report the average, maximum, and minimum CPU time comparison and acceptance rates of Type 1 users for each method in Table 5.4.

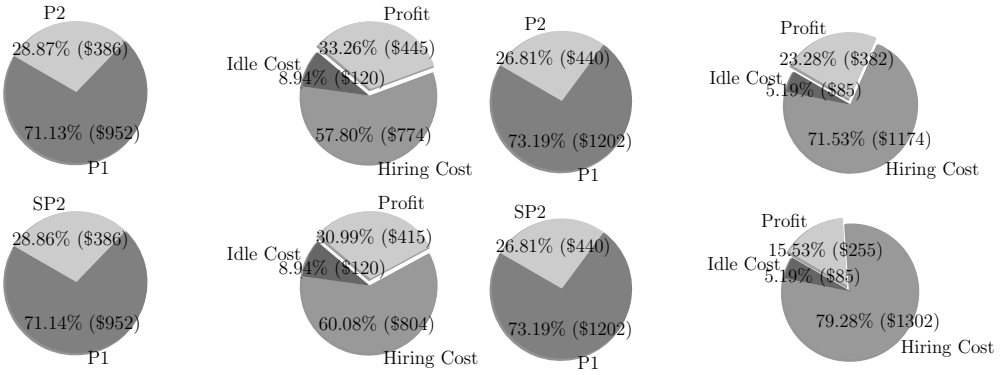
Table 5.4: CPU time (in seconds) and results comparison of the two-phase method and integrated model

			Time (Two-Phase)			Time (Integrated)			Type 1 Accept.		
K	$\|L\|$	$\|J\|$	max	min	avg	max	min	avg	max	min	avg
20	40	40	2.20	1.57	1.92	3.06	1.70	2.50	98%	83%	92%
20	40	60	59.04	5.79	23.06	65.81	7.43	26.63	100%	83%	91%
20	40	80	176.67	18.06	76.67	190.89	15.91	98.87	100%	95%	99%
30	60	40	18.60	5.37	9.67	27.52	6.45	12.49	100%	88%	93%
30	60	60	31.36	12.24	20.53	82.17	14.82	31.73	97%	90%	94%
30	60	80	483.96	33.93	256.98	1606.20	29.36	505.05	97%	82%	90%

In Table 5.4, the CPU time increases 20%-200% on average when switching from the two-phase method to directly solving the integrated model. For the instance with 60 Type 1 drivers and 80 Type 2 users, the computational time can almost reach 30 minutes. Also, we note that the solution time increases as the average acceptance rate of Type 1 drivers decreases. Moreover, the solutions obtained from the two-phase approach and the integrated model are not significantly different for most of the instances we tested.

5.5.2.4 Revenue and Cost

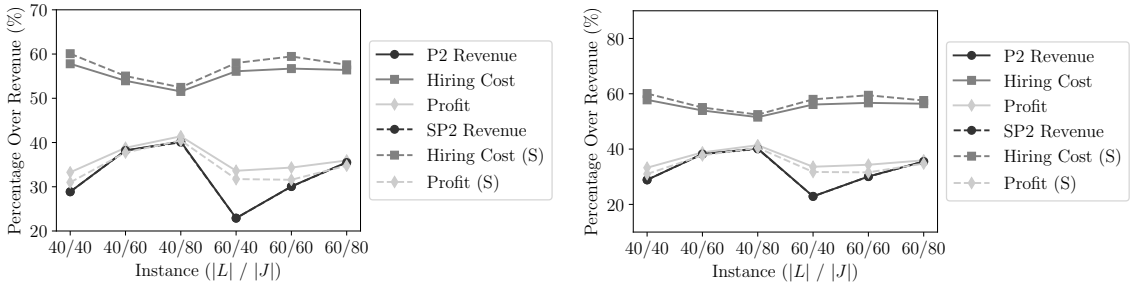
We first show the revenue and cost composition for a given instance and then extend the discussion across all instances. Figure 5.3 depicts the revenue and cost composition generated by the proposed models for the instance with $|L| = 40, |J| = 40$ for both "regular drivers" (Figure 5.3a) and "flexible drivers" (Figure 5.3b). In each figure, the top row depicts the case where we only consider deterministic traveling time, and the bottom row demonstrates the results when stochastic traveling time is taken into account. The amounts of revenue generated from serving Type 2 users are similar under P2 and SP2. However, as we consider the variability of traveling time, SP2 utilizes slightly more drivers to provide reliable service (and correspondingly, the hiring cost proportion increases by 1% and 6%, for the two cases, respectively). Comparing Figure 5.3a with Figure 5.3b, the total revenue increases in the latter case from \$1,338 to \$1,642. At the same time, the hiring cost to serve Type 2 users also increases considerably whereas the profit proportion drops.



(a) Case with "regular drivers' (b) Case with "flexible drivers"

Figure 5.3: Average revenue and cost compositions for instance with $|L| = 40, |J| = 40$ by proposed models.

Figure 5.4 depicts the changes in revenue and cost compositions across all test instances. Both cases show similar effects: given fixed $|L|$, the proportion for the revenue from serving Type 2 users and also the overall profit increase as $|J|$ increases. However, the proportion of hiring cost for Type 1 drivers decreases as more Type 2 requests appear.



(a) Case with "regular drivers"

(b) Case with "flexible drivers"

Figure 5.4: Average revenue and cost percentage for all instances by proposed models

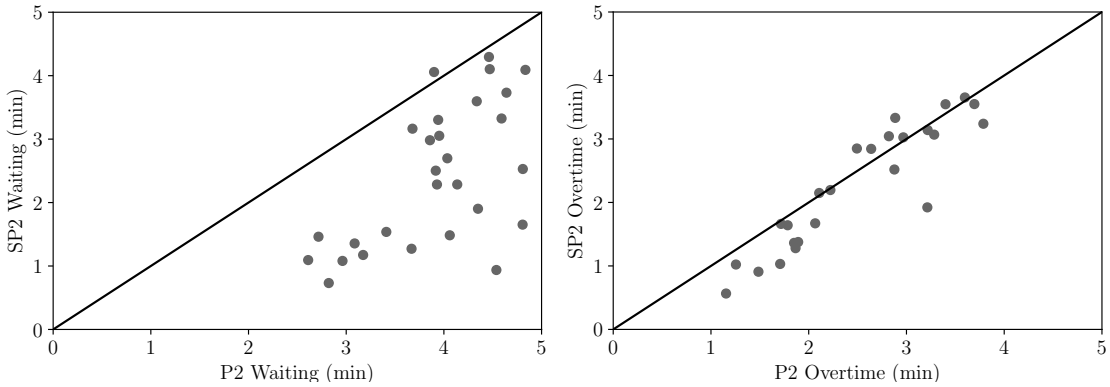
Therefore, we infer that our system produces efficient matching and scheduling to accommodate more Type 2 users. Relatively speaking, the proportion of hiring cost is higher for SP2 than P2 and the proportion of profit is lower for SP2 than P2. The finding suggests that we need to sacrifice some profit to compensate for a higher quality of scheduling service (which will be further shown in Section 5.5.2.5). Overall, the proposed system is in good financial health.

5.5.2.5 Out-of-Sample Tests

We conduct out-of-sample tests to evaluate the proposed models by measuring Type 2 users' waiting and Type 1 drivers' overtime. We generate the set of out-of-sample test scenarios Ω' following the same distribution of stochastic travel time discussed in Section 5.5.1 and use $|\Omega'| = 1000$.

After performing out-of-sample testing, let $s(l)$ be the scheduled time to return the car by Type 1 driver $l \in L$ and $s'(j)$ be the scheduled starting service time for Type 2 user $j \in J$. Values $s(l),\ l \in L$, and $s'(j),\ j \in J$, can be obtained from the optimal values of α-variables in P2 and SP2. Let $t(l,\ \omega)$ be the actual time of returning the car by Type 1 driver $l \in L$ in $\omega \in \Omega'$ and $t'(j,\ \omega)$ be the actual service starting time for Type 2 user $j \in J$ in $\omega \in \Omega'$, which will be computed based on the routing sequence and random travel and service time in scenario $\omega \in \Omega'$. Then for each test scenario ω, we calculate and report:

- wait$(j, \omega) = \max\{0,\ t(j,\ \omega) - s(j)\}$ for $j \in J$.
- overtime$(l, \omega) = \max\{0,\ t'(l,\ \omega) - s'(l)\}$ for $l \in L$.

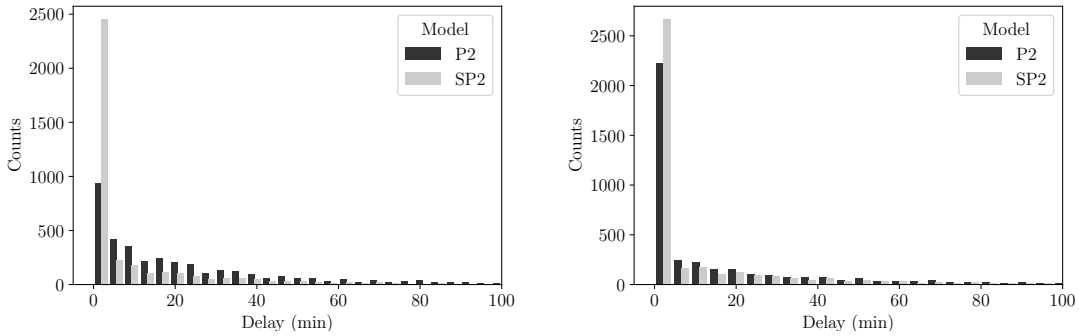


(a) Average waiting time (b) Average overtime

Figure 5.5: Average waiting time and overtime per user in proposed system

Figures 5.5a and 5.5b demonstrate the out-of-sample test results on average waiting time per Type 2 user and average overtime per Type 1 driver by proposed models. We report the results for all test instances and replications. In Figure 5.5a, for P2, the average waiting time ranges from 2.5 to 5 minutes per customer, and reduces to 1 to 4 minutes for SP2. For most of the cases, the average waiting time decreases significantly as the dots appear far below the 45-degree line. However, the performance of models for average overtime per Type 1 driver is mixed. In Figure 5.5b, the average overtime per driver ranges from 1 to 4 minutes for both models and the dots are lying around the 45-degree line, which indicates the slightly better performance of SP2. The reason that we do not see an obvious advantage of SP2 in the overtime performance is that we set the same penalty coefficients for waiting time and overtime. It leads the model to treat waiting time and overtime indifferently and therefore focus on minimizing the overall penalties. If stakeholders put more weight on overtime, one can adjust the unit penalty coefficients accordingly.

We also demonstrate the detailed out-of-sample performance for one specific instance. Figure 5.6 shows partial distributions of the average waiting time and overtime per user



(a) Average waiting time

(b) Average overtime

Figure 5.6: 90%-Tail distributions of average waiting time and overtime per user for instance with $|L| = 40$ and $|J| = 60$

in one test instance with 40 Type 1 drivers and 60 Type 2 users. Due to the significant frequency of zero waiting time and overtime per user (in more than 90% of all the out-of-sample scenarios), we report the tail distributions of the highest 10% waiting time and overtime outcomes. Both Figure 5.6a and Figure 5.6b show long-tail effect of the waiting time and overtime concentrated on small values for most cases. Comparing the performance of the P2 and SP2, SP2 yields relatively shorter waiting time and overtime than P2. Similar observations can be drawn in other instances as well.

5.6 Concluding Remarks

In this chapter, we designed a new shared mobility system to serve transportation needs of underserved populations. We integrated both carsharing and ride-hailing, and developed a two-phase approach to design and operate such a system. We evaluated the models on various instances based on synthetic data in Washtenaw County, Michigan, focusing on serving jobless, elderly, and disabled populations. Numerical results indicated the computational efficiency of our proposed solution approaches. Furthermore, the quality of service of the system was maintained at high levels.

We further extended the basic model to a two-stage stochastic programming model to

capture the randomness of vehicle travel time and service time. Numerical comparisons with a deterministic counterpart using expected values of the random parameters showed the advantages of our models. Both in-sample and out-of-sample test results demonstrated the effectiveness of matching and scheduling using our approach, where the risk of waiting and overtime both decreased.

For future research, our models and results can be extended as follows. First, in this chapter, we minimize the expected cost of overtime and waiting time. Instead, a robust optimization model that focuses on the worst-case analysis can be used for ensuring reliable operations. Second, we will collaborate with policy makers and social workers for real-world deployment of the CRS system. We aim to make more transportation data of underserved communities available to the public through further investigation of this topic.

CHAPTER 6

Conclusion

In this dissertation work, we focused on applying Operations Research techniques to address operational challenges in some applications of mobility and service sharing. Although the solution approaches and algorithms were developed for specific applications, some of them can be generalized to solve other problems. For example, the random coloring algorithm introduced in Chapter 3 can be extended to solve any VRPs. Moreover, through extensive computational experiments, we verified the reliability of the proposed models and solution approaches and showed their capability of solving mobility and service sharing problems with practical instance size. In Chapter 2, we observed that the proposed approximation algorithm was much more efficient than the general integer programming method to solve a hard VRP with compatibility constraints while providing a solution very close to the optimal one, despite its theoretical bound being large. In Chapter 3, our results demonstrated that the random coloring idea could be applied atop existing combinatorial algorithms to achieve better efficiency, and therefore, could be applied to solve instances with practical size in real-world applications. In Chapter 4, we proposed a model on a spatial-temporal network to model carsharing location design problems, and our results demonstrated the impact of having a mixed fleet on the revenue and environment. In Chapter 5, we designed a new shared mobility system combining both carsharing and ride-hailing and applied stochastic programming to reduce passengers' waiting and system overtime. Our numerical results verified the design of the system and the effectiveness of

stochastic programming.

For future research, we plan to investigate the potential improvements of techniques and solution approaches developed in this dissertation and expand their usage to other applications in mobility and service sharing. We anticipate addressing more real-world challenges in mobility and service sharing.

APPENDIX A

Appendix for Chapter 3

A.1 Extended Numerical Results

This appendix summarizes the numerical results for column generation approach on VR-PUD with different vehicle capacities Q. We consider both pulse algorithm and the random coloring algorithm as the pricing algorithm for the column generation approach. Both algorithms have been implemented in parallel using 40 computer threads. Tables A.1–A.6 summarize the results for modified Solomon and Gehring & Homerberger's benchmark. For instances with a number of customers $|V| \leq 100$, we use the first $|V| + 1$ nodes from Solomon's instance and for $|V| > 100$, we use the first $|V| + 1$ nodes from Gehring & Homberger's instances. Tables A.7 and A.8 summarize the results for modified unitary CVRP X-instances with $Q = 5$ and $Q = 6$. For both algorithms, we report the number of iterations to solve the column generation (nIter), number of columns generated in the column generation (nColumns), lower bound from the linear relaxation of MP (LB), upper bound of MP from the integral solution using the columns generated (UB), the optimality gap (Gap) computed as $\frac{UB-LB}{LB} \times 100\%$, and the runtime of the algorithm to solve the LB and UB. We set the time limit to solve the UB as 1 hour.

Table A.1: Numerical results for proposed algorithms for Type C instance in parallel implementation ($Q = 5$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	52	1178	578.32	620.50	7.29%	1.06	1.91	18	2384	578.32	634.40	9.70%	8.51	4.22
61	63	1550	753.49	795.20	5.54%	1.62	2.78	21	2793	753.49	790.90	4.97%	2.76	2.88
71	64	1514	921.21	982.10	6.61%	2.01	3.63	23	3413	921.21	981.30	6.52%	4.10	6.52
81	72	1826	1053.77	1133.00	7.52%	3.23	3.25	25	3684	1053.77	1171.00	11.13%	5.72	29.68
91	87	2121	1195.07	1277.50	6.90%	4.35	9.13	28	4180	1195.07	1265.70	5.91%	8.60	8.67
101	103	2590	1363.49	1436.00	5.32%	5.62	5.73	29	4193	1363.49	1436.70	5.37%	11.86	6.85
111	119	3041	2640.04	2783.50	5.43%	8.36	13.22	35	4983	2640.05	2814.60	6.61%	17.49	12.10
121	131	3350	2880.04	3187.10	10.66%	10.44	69.03	37	5171	2880.04	3143.10	9.13%	21.42	50.12
131	142	3741	3129.83	3391.00	8.34%	12.99	330.62	41	5655	3129.83	3311.90	5.82%	27.29	27.46
141	154	4133	3356.87	3608.20	7.49%	16.38	81.66	42	6115	3356.87	3577.80	6.58%	32.45	40.44
151	160	4270	3593.19	3826.20	6.48%	18.98	32.36	46	6664	3593.19	3840.90	6.89%	39.30	32.68

Table A.2: Numerical results for proposed algorithms for Type R instance in parallel implementation ($Q = 5$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	46	762	793.62	807.60	1.76%	0.88	0.36	18	2623	793.62	822.40	3.63%	2.06	2.54
61	57	1055	892.59	925.90	3.73%	1.43	1.71	23	3102	892.59	910.00	1.95%	3.75	3.38
71	63	1242	1067.31	1081.90	1.37%	2.00	1.24	23	3251	1067.31	1095.30	2.62%	5.22	4.28
81	77	1455	1178.48	1197.90	1.65%	3.06	1.40	28	3996	1178.48	1193.80	1.30%	8.13	5.37
91	85	1752	1290.78	1330.00	3.04%	4.11	4.80	29	4126	1290.78	1302.00	0.87%	10.65	5.11
101	94	1907	1373.71	1415.10	3.01%	5.39	5.51	33	4864	1373.71	1402.00	2.06%	14.77	9.88
111	110	2467	2972.77	3059.90	2.93%	7.55	15.14	39	5192	2972.77	3039.90	2.26%	21.01	10.28
121	126	2892	3191.53	3258.10	2.09%	9.82	7.09	40	5795	3191.53	3280.00	2.77%	24.58	18.89
131	137	3156	3450.64	3548.00	2.82%	12.55	21.59	44	6142	3450.64	3562.20	3.23%	32.86	27.99
141	142	3305	3661.81	3734.30	1.98%	14.18	6.39	45	6237	3661.81	3761.40	2.72%	34.61	21.76
151	158	3610	3882.23	3952.10	1.80%	18.15	9.96	51	6810	3882.23	3972.90	2.34%	43.83	34.39

Table A.3: Numerical results for proposed algorithms for Type RC instance in parallel implementation ($Q = 5$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	50	962	922.50	922.50	0.00%	0.99	0.48	17	2044	922.50	922.50	0.00%	1.87	1.41
61	56	985	1122.49	1224.90	9.12%	1.36	9.11	23	2752	1122.49	1223.50	9.00%	4.11	21.89
71	73	1415	1248.94	1290.40	3.32%	2.28	2.50	25	3314	1248.94	1282.30	2.67%	6.26	3.58
81	83	1645	1440.60	1440.60	0.00%	3.40	1.23	30	3718	1440.60	1440.60	0.00%	9.53	3.70
91	91	1892	1566.84	1601.50	2.21%	4.26	3.91	30	4083	1566.84	1594.30	1.75%	12.27	5.45
101	92	2016	1669.53	1701.80	1.93%	5.56	4.61	34	4571	1669.53	1720.90	3.08%	16.16	8.67
111	107	2427	2910.51	2966.40	1.92%	7.19	5.86	38	5183	2910.51	2947.60	1.27%	19.05	7.59
121	126	2763	3219.57	3259.90	1.25%	9.39	5.08	45	6046	3219.57	3271.00	1.60%	25.99	10.92
131	139	3138	3496.37	3652.00	4.45%	12.20	158.82	46	6383	3496.37	3612.30	3.32%	32.45	63.55
141	140	3438	3669.41	3790.00	3.29%	14.39	35.85	44	6296	3669.41	3845.10	4.79%	31.91	398.21
151	155	3881	3911.87	4060.50	3.80%	18.33	115.24	46	6802	3911.87	4097.20	4.74%	37.55	464.40

Table A.4: Numerical results for proposed algorithms for Type C instance in parallel implementation ($Q = 6$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	56	1312	503.49	567.50	12.71%	1.35	3.02	25	3750	503.49	558.9	11.00%	9.05	6.92
61	75	1853	652.96	744.50	14.02%	2.64	4.90	25	4133	652.95	750.5	14.94%	11.43	28.00
71	84	2100	807.58	890.30	10.24%	3.71	6.67	31	4705	807.57	892.6	10.53%	20.69	13.98
81	98	2432	909.55	1027.90	13.01%	5.58	7.32	27	5239	909.55	1046.4	15.05%	20.99	14.03
91	110	2866	1030.55	1168.00	13.34%	7.87	10.04	34	5922	1030.55	1194.8	15.94%	32.43	90.63
101	135	3414	1176.05	1323.60	12.55%	11.45	13.11	36	6300	1176.05	1343.2	14.21%	38.48	85.78
111	162	4193	2288.64	2430.70	6.21%	15.91	24.53	43	7762	2288.64	2501.1	9.28%	68.25	68.55
121	175	4478	2474.52	2713.00	9.64%	20.89	73.83	47	8087	2474.51	2590.5	4.69%	84.65	22.02
131	172	4588	2685.52	2936.10	9.33%	24.81	102.19	47	8885	2685.52	2964.1	10.37%	97.88	81.90
141	195	5200	2868.49	3140.20	9.47%	32.76	190.50	50	8850	2868.49	3120.2	8.77%	122.91	63.05
151	216	5786	3058.15	3395.90	11.04%	39.04	349.14	54	10118	3058.15	3422.3	11.91%	147.49	525.39

Table A.5: Numerical results for proposed algorithms for Type R instance in parallel implementation ($Q = 6$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	50	888	710.13	760.00	7.02%	1.11	2.43	29	3697	710.13	741.10	4.36%	10.87	5.59
61	63	1190	798.46	823.50	3.14%	1.91	3.07	24	4506	798.46	835.20	4.60%	13.16	8.21
71	77	1449	954.20	978.10	2.50%	3.11	3.27	28	5022	954.20	973.90	2.06%	19.28	7.57
81	91	1787	1047.07	1091.10	4.21%	4.68	6.17	32	5924	1047.07	1084.10	3.54%	27.35	14.03
91	103	2245	1144.94	1201.70	4.96%	6.78	23.05	42	7108	1144.94	1178.00	2.89%	47.87	16.36
101	109	2293	1214.31	1242.80	2.35%	8.13	5.48	39	7528	1214.31	1251.50	3.06%	52.13	23.91
111	143	3235	2623.57	2691.10	2.57%	14.15	6.64	48	8241	2623.57	2685.60	2.36%	78.99	28.80
121	153	3748	2805.75	2830.70	0.89%	17.75	7.14	47	8251	2805.75	2838.10	1.15%	89.75	16.11
131	163	3958	3021.94	3135.40	3.75%	22.40	63.31	52	9576	3021.94	3112.40	2.99%	112.79	48.16
141	174	4219	3197.56	3282.00	2.64%	25.35	28.55	56	9931	3197.56	3291.40	2.93%	137.00	65.89
151	191	4664	3381.61	3492.50	3.28%	32.47	69.85	57	10668	3381.61	3489.10	3.18%	165.27	74.94

Table A.6: Numerical results for proposed algorithms for Type RC instance in parallel implementation ($Q = 6$)

nNode	Pulse							Random Coloring						
	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)	nIter	nCol	LB	UB	Gap	t_{LB} (s)	t_{UB} (s)
51	49	1000	792.20	916.90	15.74%	1.18	2.01	22	2869	792.20	912.30	15.16%	8.62	5.10
61	60	1293	968.22	1047.40	8.18%	2.02	3.82	28	3967	968.22	1037.10	7.11%	17.79	9.09
71	81	1735	1090.28	1136.20	4.21%	3.31	6.58	29	4788	1090.28	1149.50	5.43%	24.80	24.93
81	91	1944	1259.23	1298.00	3.08%	4.56	4.07	39	5847	1259.23	1301.80	3.38%	40.90	12.07
91	106	2345	1374.69	1421.10	3.38%	6.66	7.90	38	6547	1374.69	1398.90	1.76%	50.16	13.91
101	124	2744	1452.87	1498.20	3.12%	9.05	6.89	37	6749	1452.87	1533.90	5.58%	56.09	20.61
111	129	3070	2540.23	2629.60	3.52%	12.44	12.88	44	7891	2540.23	2696.10	6.14%	65.04	93.51
121	143	3435	2810.63	2870.80	2.14%	16.22	10.04	49	8392	2810.63	2896.40	3.05%	84.86	32.03
131	157	3686	3043.66	3156.20	3.70%	20.63	59.95	56	9402	3043.66	3133.20	2.94%	110.01	75.65
141	170	4147	3187.66	3329.90	4.46%	25.62	205.63	56	10568	3187.66	3335.60	4.64%	129.80	413.40
151	177	4401	3399.65	3513.20	3.34%	31.97	110.46	55	10549	3399.65	3520.30	3.55%	148.03	146.48

Table A.7: Numerical results for unitary X instances with Q=5

Instance	n	Pulse					Random Coloring				
		LB	UB	Gap	t_{LB} (min)	t_{UB} (min)	LB	UB	Gap	t_{LB} (min)	t_{UB} (min)
X-n120-k6	120	38683.88	39112	1.09%	0.27	0.02	38683.88	39096	1.05%	0.63	0.01
X-n157-k13	157	35447.29	36609	3.17%	1.02	0.04	35447.29	36508	2.91%	1.05	8.39
X-n181-k23	181	37673.14	40385	6.72%	0.74	4.81	37673.14	39064	3.56%	1.41	26.68
X-n219-k73	219	73307.44	76965	4.75%	1.73	0.17	73307.44	74278	1.31%	2.32	0.43
X-n237-k14	237	77629.8	81267	4.48%	1.97	0.35	77629.8	79338	2.15%	2.81	13.87
X-n275-k28	275	36336.56	38062	4.53%	2.74	0.70	36336.56	37583	3.32%	3.60	22.70
X-n317-k53	317	92699.78	99170	6.52%	7.01	0.07	92699.78	95654	3.09%	6.63	9.19
X-n331-k15	331	111908.6	120629	7.23%	5.69	0.38	111908.6	116081	3.59%	7.42	–
X-n376-k94	376	119578.9	128564	6.99%	8.73	1.83	119578.9	123865	3.46%	10.85	–
X-n439-k37	439	73360.37	79267	7.45%	9.23	0.34	73360.37	76305	3.86%	14.98	17.13
X-n502-k39	502	169070.4	181780	6.99%	75.85	0.89	169070.4	175107	3.45%	29.07	–
X-n548-k50	548	177270.1	187951	5.68%	32.60	5.63	177270.1	187453	5.43%	34.07	–
X-n655-k131	655	106556.7	115027	7.36%	59.26	0.25	106556.7	111769	4.66%	61.95	–
X-n801-k40	801	254711.4	278893	8.67%	105.84	3.02	254711.4	275793	7.64%	118.27	44.11
X-n856-k95	856	149414.7	166257	10.13%	98.08	0.31	149414.7	159191	6.14%	134.93	1.85
X-n957-k87	957	171141.4	186962	8.46%	154.73	0.30	171141.4	184950	7.47%	189.30	28.17

–: solution time reaches 60-minute time limit.

Table A.8: Numerical results for unitary X instances with Q=6

Instance	n	Pulse					Random Coloring				
		LB	UB	Gap	t_{LB} (min)	t_{UB} (min)	LB	UB	Gap	t_{LB} (min)	t_{UB} (min)
X-n120-k6	120	33062.43	34916	5.31%	0.42	0.31	33062.43	34789	4.96%	1.51	0.66
X-n157-k13	157	30077.32	31723	5.19%	4.12	0.01	30077.32	31033	3.08%	4.12	0.51
X-n181-k23	181	32091.36	34806	7.80%	1.30	0.41	32091.36	33392	3.90%	5.36	31.06
X-n219-k73	219	62252.87	65262	4.61%	3.45	0.02	62252.87	64478	3.45%	9.89	2.40
X-n237-k14	237	65950.46	69863	5.60%	3.32	0.75	65950.46	68202	3.30%	11.92	24.14
X-n275-k28	275	31236.96	33132	5.72%	5.60	0.94	31236.96	32455	3.75%	16.36	11.10
X-n317-k53	317	78115.76	85531	8.67%	14.39	0.09	78115.76	82953	5.83%	33.15	20.80
X-n331-k15	331	94655.77	107309	11.79%	9.25	0.47	94655.77	99853	5.20%	36.13	–
X-n376-k94	376	101162.6	113409	10.80%	15.26	3.19	101162.6	106355	4.88%	51.50	39.39
X-n439-k37	439	62675.72	68469	8.46%	21.25	0.30	62675.72	66341	5.52%	83.82	–
X-n502-k39	502	141816.1	153529	7.63%	564.11	0.15	141816.1	148973	4.80%	142.96	–
X-n548-k50	548	149497.2	164883	9.33%	66.96	0.93	149497.2	158010	5.39%	177.83	38.07
X-n655-k131	655	89912.22	98703	8.91%	192.21	1.82	89912.22	94933	5.29%	331.71	–
X-n801-k40	801	214261	236558	9.43%	205.19	0.57	214261	231453	7.43%	688.48	–
X-n856-k95	856	126628.7	140397	9.81%	288.08	0.61	126628.7	137706	8.04%	726.95	–
X-n957-k87	957	144860.1	161364	10.23%	352.01	0.41	144860.1	156071	7.18%	1152.08	–

–: solution time reaches 60-minute time limit.

BIBLIOGRAPHY

Absi, N., Dauzère-Pérès, S., Kedad-Sidhoum, S., Penz, B., and Rapine, C. (2013). Lot sizing with carbon emission constraints. *European Journal of Operational Research*, 227(1):55–61.

Achuthan, N. R., Caccetta, L., and Hill, S. P. (2003). An improved branch-and-cut algorithm for the capacitated vehicle routing problem. *Transportation Science*, 37(2):153–169.

Adaji, A., Melin, G. J., Campbell, R. L., Lohse, C. M., Westphal, J. J., and Katzelnick, D. J. (2018). Patient-centered medical home membership is associated with decreased hospital admissions for emergency department behavioral health patients. *Population Health Management*, 21(3):172–179.

Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303.

Ahmed, S. (2013). A scenario decomposition algorithm for 0–1 stochastic programs. *Operations Research Letters*, 41(6):565–569.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Application*. Prentice Hall.

Allaoua, H., Borne, S., Létocart, L., and Calvo, R. W. (2013). A matheuristic approach for solving a home health care problem. *Electronic Notes in Discrete Mathematics*, 41:471–478.

Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., and Sahinalp, S. C. (2008). Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249.

Alon, N., Yuster, R., and Zwick, U. (1995). Color-coding. *J. ACM*, 42(4):844–856.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467.

American Academy of Family Physicians (2008). Joint principles of the patient-centered medical home. *Delaware Medical Journal*, 80(1):21.

Applegate, D., Cook, W., Dash, S., and Rohe, A. (2002). Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing*, 14(2):132–143.

Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Arkin, E. M., Hassin, R., and Levin, A. (2006). Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18.

Arora, S. and Karakostas, G. (2006). A $2 + \epsilon$ approximation algorithm for the k-MST problem. *Mathematical Programming*, 107(3):491–504.

Bachouch, R. B., Guinet, A., and Hajri-Gabouj, S. (2011). A decision-making tool for home health care nurses' planning. In *Supply Chain Forum: An International Journal*, volume 12, pages 14–20. Taylor & Francis.

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, 19(6):621–636.

Baldacci, R., Bartolini, E., Mingozzi, A., and Roberti, R. (2010). An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268.

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283.

Bansal, N., Blum, A., Chawla, S., and Meyerson, A. (2004). Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 166–174. ACM.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Barrios, J. and Godier, J. (2014). Fleet sizing for flexible carsharing systems: Simulation-based approach. *Transportation Research Record: Journal of the Transportation Research Board*, 2416(1):1–9.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252.

Bettinelli, A., Ceselli, A., and Righini, G. (2011). A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 19(5):723–740.

Birge, J. R. and Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer Science & Business Media.

Blum, A., Ravi, R., and Vempala, S. (1996). A constant-factor approximation algorithm for the k MST problem. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, pages 442–448. ACM.

Boyacı, B., Zografos, K. G., and Geroliminis, N. (2015). An optimization framework for the development of efficient one-way car-sharing systems. *European Journal of Operational Research*, 240(3):718–733.

Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313.

Brandstatter, G., Leitner, M., and Ljubic, I. (2016). Locations of charging stations in electric car sharing systems. Technical report, Department of Statistics and Operations Research, University of Vienna, Vienna, Austria.

Brook, D. (2004). Carsharing–start up issues and new operational models. In *Transportation Research Board 83rd Annual Meeting, Washington, DC*. Citeseer.

Car2Go (2015). `https://www.car2go.com/`.

Carlsson, J., Ge, D., Subramaniam, A., Wu, A., and Ye, Y. (2009). Solving min-max multi-depot vehicle routing problem. *Lectures on Global Optimization*, 55:31–46.

Chalasani, P. and Motwani, R. (1999). Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28(6):2133–2149.

Chan, N. D. and Shaheen, S. A. (2012). Ridesharing in North America: Past, present, and future. *Transport Reviews*, 32(1):93–112.

Chang, J., Yu, M., Shen, S., and Xu, M. (2017). Location design and relocation of a mixed carsharing fleet with a CO_2 emission constraint. *Service Science*, 9(3):205–218.

Charikar, M., Khuller, S., and Raghavachari, B. (2001). Algorithms for capacitated vehicle routing. *SIAM Journal on Computing*, 31(3):665–682.

Chekuri, C., Korula, N., and Pál, M. (2012). Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms (TALG)*, 8(3):23.

Chekuri, C. and Kumar, A. (2004). Maximum coverage problem with group budget constraints and applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 72–83. Springer.

Chesto, J. (2015). Zipcar seeks 150 park-anywhere permits. `http://senseable.mit.edu/news/pdfs/20150224_BostonGlobe.pdf`.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282.

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235.

City Carshare (2014). Our Mission. `https://citycarshare.org/why-city-carshare/our-mission/`.

Clewlow, R. R. and Mishra, G. S. (2017). Disruptive transportation: the adoption, utilization, and impacts of ride-hailing in the united states. *University of California, Davis, Institute of Transportation Studies, Davis, CA, Research Report UCD-ITS-RR-17-07*.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research*, 153(1):29–46.

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

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

Daskin, M. S. (2011). *Network and Discrete Location: Models, Algorithms, and Applications*. John Wiley & Sons.

Daskin, M. S., Coullard, C. R., and Shen, Z.-J. M. (2002). An inventory-location model: Formulation, solution algorithm and computational results. *Annals of Operations Research*, 110(1-4):83–106.

Daskin, M. S., Snyder, L. V., and Berger, R. T. (2005). Facility location in supply chain design. In *Logistics systems: Design and optimization*, pages 39–65. Springer.

de Almeida Correia, G. H. and Antunes, A. P. (2012). Optimization approach to depot location and trip selection in one-way carsharing systems. *Transportation Research Part E: Logistics and Transportation Review*, 48(1):233–247.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). *Column Generation*. Springer Science & Business Media.

Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). Time constrained routing and scheduling. *Handbooks in Operations Research and Management Science*, 8:35–139.

Dinur, I. and Steurer, D. (2014). Analytical approach to parallel repetition. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, pages 624–633. ACM.

Dohn, A., Kolind, E., and Clausen, J. (2009). The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4):1145–1157.

Downey, R. G. and Fellows, M. R. (2012). *Parameterized Complexity*. Springer Science & Business Media.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978.

Duque, D., Lozano, L., and Medaglia, A. L. (2015). Solving the orienteering problem with time windows via the pulse framework. *Computers & Operations Research*, 54:168–176.

Eveborn, P., Flisberg, P., and Rönnqvist, M. (2006). Laps care—an operational system for staff planning of home care. *European Journal of Operational Research*, 171(3):962–976.

Even, G., Garg, N., Könemann, J., Ravi, R., and Sinha, A. (2004). Min–max tree covers of graphs. *Operations Research Letters*, 32(4):309–315.

Fan, W. D. (2014). Optimizing strategic allocation of vehicles for one-way car-sharing systems under demand uncertainty. *Journal of the Transportation Research Forum*, 53(3):7–20.

Febbraro, A., Sacco, N., and Saeednia, M. (2012). One-way carsharing: Solving the relocation problem. *Transportation Research Record: Journal of the Transportation Research Board*, 2319(1):113–120.

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

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR: Information Systems and Operational Research*, 45(4):239–256.

Fernandez, J., Blacking, J., Dundes, A., Edmonson, M. S., Etzkorn, K. P., Haydu, G. G., Kearney, M., Kehoe, A. B., Loveland, F., McCormack, W. C., et al. (1974). The mission of metaphor in expressive culture. *Current Anthropology*, pages 119–145.

Fikar, C. and Hirsch, P. (2017). Home health care routing and scheduling: A review. *Computers & Operations Research*, 77:86–95.

Firth, S. (2016). Q&A with Detroit's new director of public health. `https://www.medpagetoday.com/publichealthpolicy/publichealth/55883`.

Fukasawa, R., He, Q., and Song, Y. (2015). A branch-cut-and-price algorithm for the energy minimization vehicle routing problem. *Transportation Science*, 50(1):23–34.

Fukasawa, R., Longo, H., Lysgaard, J., de Aragão, M. P., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511.

Garg, N. (1996). A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, pages 302–309, Washington, DC, USA. IEEE Computer Society.

Garg, N. (2005). Saving an epsilon: A 2-approximation for the k-MST problem in graphs. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, pages 396–402. ACM.

Golden, B. L., Laporte, G., and Taillard, É. D. (1997). An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. *Computers & Operations Research*, 24(5):445–452.

Golden, B. L., Levy, L., and Vohra, R. (1987). The orienteering problem. *Naval Research Logistics*, 34(3):307–318.

Golden, B. L., Raghavan, S., and Wasil, E. A. (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Science & Business Media.

Gørtz, I. L., Molinaro, M., Nagarajan, V., and Ravi, R. (2016). Capacitated vehicle routing with nonuniform speeds. *Mathematics of Operations Research*, 41(1):318–331.

Gross, M. B., Hogarth, J. M., Schmeiser, M. D., et al. (2012). Use of financial services by the unbanked and underbanked and the potential for mobile financial services adoption. *Federal Reserve Bulletin*, 98(4):1–20.

Harris-Kojetin, L., Sengupta, M., Park-Lee, E., and Valverde, R. (2013). Long-term care services in the united states: 2013 overview. *Vital & health statistics. Series 3, Analytical and epidemiological studies/[US Dept. of Health and Human Services, Public Health Service, National Center for Health Statistics]*, 3(37):1–107.

He, L., Mak, H. Y., Rong, Y., and Shen, Z. J. M. (2016). Service region design for urban electric vehicle sharing systems. *Manufacturing & Service Operations Management*, 19(2):309–327.

Irnich, S. and Villeneuve, D. (2006). The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, 18(3):391–406.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511.

Kariv, O. and Hakimi, S. L. (1979). An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538.

Kek, A. G. H., Cheu, R. L., Meng, Q., and Fung, C. H. (2009). A decision support system for vehicle relocation operations in carsharing systems. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):149–158.

Klein, P. N. and Ravi, R. (1995). A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115.

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

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

Lanzarone, E. and Matta, A. (2014). Robust nurse-to-patient assignment in home care services to minimize overtimes under continuity of care. *Operations Research for Health Care*, 3(2):48–58.

Laporte, G., Louveaux, F., and Mercure, H. (1992). The vehicle routing problem with stochastic travel times. *Transportation Science*, 26(3):161–170.

Laporte, G. and Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations research letters*, 13(3):133–142.

Laporte, G., Louveaux, F. V., and Van Hamme, L. (2002). An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research*, 50(3):415–423.

Laporte, G., Meunier, F., and Calvo, R. W. (2015). Shared mobility systems. *4OR*, 13(4):341–360.

Letchford, A. N., Eglese, R. W., and Lysgaard, J. (2002). Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming*, 94(1):21–40.

Lozano, L., Duque, D., and Medaglia, A. L. (2015). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384.

Lu, M., Shen, S., and Chen, Z. (2018). Optimizing the profitability and quality of service in car-share systems under demand uncertainty. *Manufacturing and Service Operations Management*, 20(2).

Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445.

Maass, K. L., Daskin, M. S., and Shen, S. (2016). Mitigating hard capacity constraints with inventory in facility location modeling. *IIE Transactions*, 48(2):120–133.

Magnanti, T. L. and Wong, R. T. (1984). Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55.

Malandraki, C. and Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200.

Malik, W., Rathinam, S., and Darbha, S. (2007). An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Operations Research Letters*, 35(6):747–753.

Martin, C. J. (2016). The sharing economy: A pathway to sustainability or a nightmarish form of neoliberal capitalism? *Ecological Economics*, 121:149–159.

Martin, C. S. and Salavatipour, M. R. (2017). Minimizing latency of capacitated k-tours. *Algorithmica*, pages 1–20.

Martin, E. W. and Shaheen, S. A. (2011). Greenhouse gas emission impacts of carsharing in North America. *Intelligent Transportation Systems, IEEE Transactions on*, 12(4):1074–1086.

Martinez, L. M., Caetano, L., Eiró, T., and Cruz, F. (2012). An optimisation algorithm to establish the location of stations of a mixed fleet biking system: An application to the City of Lisbon. *Procedia-Social and Behavioral Sciences*, 54:513–524.

McLaughlin, K. (2015). Detroit has become such an 'economic desert' that residents now commute up to four-hours each day just to find work. Daily Mail. `http://www.dailymail.co.uk/news/article-2975237/Dearth-jobs-barrier-post-bankruptcy-Detroits-growth.html`.

Melkote, S. and Daskin, M. S. (2001). An integrated model of facility location and transportation network design. *Transportation Research Part A: Policy and Practice*, 35(6):515–538.

Melo, M. T., Nickel, S., and Saldanha-da Gama, F. (2009). Facility location and supply chain management–A review. *European Journal of Operational Research*, 196(2):401–412.

Millard-Ball, A. (2005). *Car-Sharing: Where and How it Succeeds*, volume 108. Transportation Research Board.

Muheim, P. and Reinhardt, E. (1999). Carsharing: the key to combined mobility. *World Transport Policy & Practice*, 5(3).

Musich, S., Wang, S. S., Hawkins, K., and Yeh, C. S. (2015). Homebound older adults: Prevalence, characteristics, health care utilization and quality of care. *Geriatric Nursing*, 36(6):445–450.

Nair, R. and Miller-Hooks, E. (2011). Fleet management for vehicle sharing operations. *Transportation Science*, 45(4):524–540.

Nair, R. and Miller-Hooks, E. (2014). Equilibrium network design of shared-vehicle systems. *European Journal of Operational Research*, 235(1):47–61.

Navigant Research (2017). Mobility as a service the future of moving people: Carsharing, ride-hailing, micro transit, automated mobility, and p2p rental services. Technical report, Navigant Research.

Nourinejad, M. and Roorda, M. J. (2014). A dynamic carsharing decision support system. *Transportation Research Part E: Logistics and Transportation Review*, 66:36–50.

Owen, S. H. and Daskin, M. S. (1998). Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423–447.

Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017a). New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017b). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100.

Poggi de Aragao, M. and Uchoa, E. (2003). Integer program reformulation for robust branch-and-cut-and-price algorithms. In *In Proceedings of the Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*. Citeseer.

Polus, A. (1979). A study of travel time and reliability on arterial routes. *Transportation*, 8(2):141–151.

Ralphs, T. K., Kopman, L., Pulleyblank, W. R., and Trotter, L. E. (2003). On the capacitated vehicle routing problem. *Mathematical Programming*, 94(2-3):343–359.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273.

Salmond, S. and Ropis, P. E. (2005). Job stress and general well-being: A comparative study of medical-surgical and home care nurses. *Medsurg Nursing*, 14(5):301.

Shaheen, S., Chan, N., Bansal, A., and Cohen, A. (2015). Shared mobility: A sustainability & technologies workshop: definitions, industry developments, and early understanding. Technical report, Innovative Mobility Research.

Shaheen, S. A. and Cohen, A. P. (2014). Innovative mobility carsharing outlook. Technical report, Transportation Sustainability Research Institute - University of California, Berkeley.

Shapiro, A., Dentcheva, D., and Ruszczyński, A. (2014). *Lectures on stochastic programming: modeling and theory*. SIAM.

Shapiro, A. and Philpott, A. (2007). A tutorial on stochastic programming.

Shen, Z.-J. M., Coullard, C., and Daskin, M. S. (2003). A joint location-inventory model. *Transportation Science*, 37(1):40–55.

Shen, Z.-J. M., Zhan, R. L., and Zhang, J. (2011). The reliable facility location problem: Formulations, heuristics, and approximation algorithms. *INFORMS Journal on Computing*, 23(3):470–482.

Smith, A. (2016). Shared, collaborative and on demand: The new digital economy. *Pew Research Center*, 19.

Snyder, L. V. (2006). Facility location under uncertainty: A review. *IIE Transactions*, 38(7):547–564.

Snyder, L. V., Daskin, M. S., and Teo, C.-P. (2007). The stochastic location model with risk pooling. *European Journal of Operational Research*, 179(3):1221–1238.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Svitkina, Z. and Tardos, É. (2004). Min-max multiway cut. In *7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX*, pages 207–218.

Taş, D., Dellaert, N., Van Woensel, T., and De Kok, T. (2013). Vehicle routing problem with stochastic travel times including soft time windows and service costs. *Computers & Operations Research*, 40(1):214–224.

The National Association for Home Care & Hospice (2010). Basic statistics about home care. Technical report, The National Association for Home Care & Hospice.

Thebault-Spieker, J., Terveen, L. G., and Hecht, B. (2015). Avoiding the south side and the suburbs: The geography of mobile crowdsourcing markets. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 265–275. ACM.

Toth, P. and Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications*, volume 18. SIAM.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858.

United States Census Bureau (2010). National census tracts gazetteer. `https://www.census.gov/geo/maps-data/data/gazetteer2010.html`.

United States Environmental Protection Agency (2013). Sources of greenhouse gas emissions. Technical report, United States Environmental Protection Agency.

Weikl, S. and Bogenberger, K. (2013). Relocation strategies and algorithms for free-floating car sharing systems. *Intelligent Transportation Systems Magazine, IEEE*, 5(4):100–111.

Williamson, D. P. and Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge university press.

Yu, M., Nagarajan, V., and Shen, S. (2017). Minimum makespan vehicle routing problem with compatibility constraints. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 244–253. Springer.

Yu, M., Nagarajan, V., and Shen, S. (2018). An approximation algorithm for vehicle routing with compatibility constraints. *Operations Research Letters*, 46(6):579–584.

Yu, M. and Shen, S. (2020). An integrated car-and-ride sharing system for mobilizing heterogeneous travelers with application in underserved communities. *IISE Transactions*, 52(2):151–165.

Zhan, Y., Wang, Z., and Wan, G. (2015). Home care routing and appointment scheduling with stochastic service durations. *Available at SSRN*.

Zhang, Y., Lu, M., and Shen, S. (2018). On the values of vehicle-to-grid electricity selling in electric vehicle sharing. *to appear in Manufacturing and Service Operations Management*.

Zipcar (2015). https://zipcar.com.