Copyright Warning

Use of this thesis/dissertation/project is for the purpose of private study or scholarly research only. *Users must comply with the Copyright Ordinance.*

Anyone who consults this thesis/dissertation/project is understood to recognise that its copyright rests with its author and that no part of it may be reproduced without the author’s prior written consent.
Copyright Warning

Use of this thesis/dissertation/project is for the purpose of private study or scholarly research only. *Users must comply with the Copyright Ordinance.*

Anyone who consults this thesis/dissertation/project is understood to recognise that its copyright rests with its author and that no part of it may be reproduced without the author’s prior written consent.
CITY UNIVERSITY OF HONG KONG
香港城市大學

Vehicle Routing Problems with Cumulative Cost Structure
帶累積成本結構的車輛路徑問題

Submitted to
Department of Management Sciences
管理科學系
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
哲學博士學位

by

LUO Zhixing
羅志興

June 2014
二零一四年六月
Vehicle Routing Problems with Cumulative Cost Structure

by LUO Zhixing
Department of Management Sciences
College of Business
City University of Hong Kong

Abstract

This thesis studies a class of vehicle routing problems with cumulative cost structure (VRPCCS), which have wide applications in practice. Vehicle routing problem (VRP) has been a hot research area in the field of operations research for more than five decades. It consists of finding a set of routes for a fleet of vehicles with known capacities to service a given set of customers while minimizing the total travel cost and satisfying various constraints, such as capacity constraint, time window constraints and maximum travel distance constraints. Unlike the classical cost structure where the travel cost is only proportional to the travel distance, the cumulative cost structure defines the travel cost of vehicles per unit distance as a function of another quantity (e.g., number of customers served, weight of vehicle) which accumulates as the vehicle travels along the route. From the perspective of modeling, the VRPCCS is a generalization of the classical VRP.

Compared to the classical VRP, the VRPCCS has received increasing attention in the literature in recent years due to its applicability to many distribution systems. Some cumulative cost structures, e.g. the customer waiting time, are considered in the customer-centric or service-based objective functions for increasing the level of customers’ satisfaction. Another interesting example of the cumulative cost structure arises in Chinese expressway system where expressway tolls are levied according to vehicle weight and traveling distance, which is
referred to as *toll-by-weight scheme*.

The contributions of this thesis are fourfold. Firstly, we introduce three new and practical vehicle routing problems that take the cumulative cost structures into account to the literature. Secondly, we formulate these new problems into different types of mathematical programming models and conduct detailed and comprehensive analysis on their properties. Thirdly, based on the models and the properties we propose effective solution procedures for these problems, both in exact and heuristic ways. Lastly, we provide a large number of benchmark instances as well as detailed solution results, which facilitate the future researchers to investigate these or related problems.

The first problem studied in this thesis is the multiple traveling repairmen problem with distance constraints (MTRPD), which is an extension of the multiple traveling repairman problem by considering a limitation on the total distance that a vehicle can travel. In the MTRPD, a fleet of vehicles is dispatched to serve a set of customers. Each vehicle that starts from and ends at the depot is not allowed to travel a distance longer than a predetermined limit and each customer must be visited exactly once. The objective is to minimize the total waiting time of all customers after the vehicles leave the depot. To optimally solve the MTRPD, we have implemented three branch-and-price-and-cut algorithms, which correspond to three types of label-setting algorithms applied to the pricing subproblem. Experiments show that the branch-and-price-and-cut algorithm that includes the bounded bi-directional label-setting algorithm and space state relaxation outperforms the other two algorithms, and is able to solve most of the test instances optimally.

The second problem is called the split-collection vehicle routing problem with time windows and linear weight-related cost (SCVRPTWL), which is a new VRP variant that simultaneously considers time windows, split collection and linear
weight-related transportation cost. This problem consists of determining least-cost vehicle routes to serve a set of customers while respecting the restrictions of vehicle capacity and time windows. The travel cost per unit distance is a linear function of the vehicle weight and the customer demand can be fulfilled by multiple vehicles. The SCVRPTWL can be viewed as a generalization of the classic split-delivery vehicle routing problem with time windows (SDVRPTW). To solve this problem, we propose an exact branch-and-price-and-cut algorithm, where the pricing subproblem is a resource-constrained elementary least-cost path problem. We first prove that at least an optimal solution to the pricing subproblem is associated with an extreme collection pattern, and then design a tailored label-setting algorithm to solve it. Computational results show that our proposed algorithm can handle both the SDVRPTW and our problem efficiently.

The third problem is a VRP variant that incorporates stochastic demands and toll-by-weight scheme, which is therefore named the vehicle routing problem with stochastic demands and toll-by-weight scheme (VRPSD-TBW). We deal with this problem using a priori optimization solution strategy with dynamic recourse, where the key of the solution procedure is to compute the expected cost of a given route. The problem aims to design a set of collection routes with minimal total expected travel cost to fulfill the requests of a set of customers. In order to find the set of best vehicle routes, we propose an adaptive large neighborhood search (ALNS) algorithm that employs several approximate evaluation schemes for the expected cost of the route and several removal and insertion heuristics. Experiments on a set of benchmark instances which are generated based on the information from the real data in several Chinese provinces demonstrate the effectiveness of our ALNS algorithm.
CITY UNIVERSITY OF HONG KONG
Qualifying Panel and Examination Panel

Surname: LUO
First Name: Zhixing
Degree: Doctor of Philosophy
College/Department: Department of Management Sciences

The Qualifying Panel of the above student is composed of:

*Supervisor(s)*
Prof. LIM Leong Chye Andrew Department of Management Sciences
City University of Hong Kong

*Qualifying Panel Members(s)*
Dr. HE Simai Department of Management Sciences
City University of Hong Kong
Dr. LEUNG Chi Hang Sephen Department of Management Sciences
City University of Hong Kong

This thesis has been examined and approved by the following examiners:

Dr. SHUM Stephen Wan Hang Department of Management Sciences
City University of Hong Kong
Prof. LAI Kin Keung Department of Management Sciences
City University of Hong Kong
Prof. LIM Leong Chye Andrew Department of Management Sciences
City University of Hong Kong
Prof. SIM Melvyn Department of Decision Sciences
National University of Singapore
Acknowledgments

Foremost, I would like to thank my supervisor Prof. Andrew Lim for giving me the opportunity to peruse the doctoral degree in CityU, and his continuous guidance and support over the last four years. He patiently provided the vision, encouragement and advise necessary for me to proceed through the doctoral program and complete this thesis.

I am grateful to Prof. LAI Kin Keung, Dr. Shum Stephen Wan Hang and Prof. Melvyn Sim for serving on my thesis committee.

I would like to offer my special thanks to Prof. Chen Youhua Frank, Prof. Yan Houmin, Prof. Wan Tze-Kin Alan, Dr. Li Yanzhi, Dr. He Simai, Dr. Liu Guangwu, Dr. Lu Ye, Dr. Shou Biying, Dr. Yu Yimin and other faculty members for their instruction and assistance during my Ph.D. career.

I am particularly grateful for the assistance and guidance given by Dr. Qin Hu and Dr. Zhu Wenbin in the last four years.

Many thanks also to all my classmates and teammates: Wei Lijun, Zhang Bingfeng, Zhang Zhenzhen, Zhang Zizhen, Chen Yibo, Yan Tao, Wang Ning, Liu Mengyang, Li Chongshou, Liu Fan, Tian Tian, Jin Bo, Hu Qian, Gong Lijun, Xue Li, Shen Huaxiao, Cui Ligang, Liu Tian and Che Chan Hou for the help and happy time spent together.

Finally, and most importantly, I would like to thank my family, for their unconditional love and support.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Qualifying Panel and Examination Panel</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xii</td>
</tr>
</tbody>
</table>

### 1 Introduction

1.1 Background and Motivation                                             1
1.2 Contribution                                                          6
1.3 Thesis Organization                                                    7

### 2 Branch-and-Price-and-Cut for the Multiple Traveling Repairman Problem with Distance Constraints

2.1 Introduction                                                          8
2.2 Mathematical formulation                                              12
2.3 Column generation                                                     16
2.3.1 The pricing subproblem                                              17
2.3.2 The label-setting algorithm                                          19
2.3.3 Accelerating strategies ............................................. 21

2.4 Branch-and-price-and-cut algorithm ................................ 25
  2.4.1 Initial columns and upper bound .................................. 26
  2.4.2 Separation algorithm ................................................ 26
  2.4.3 Search strategy ...................................................... 27
  2.4.4 Branching strategies ............................................... 27

2.5 Computational experiments ............................................ 28
  2.5.1 Instance generation ................................................ 29
  2.5.2 Experimental setup ................................................. 29
  2.5.3 Results and analysis ............................................... 30

2.6 Conclusions ............................................................ 35

3 Branch-and-price-and-cut for the Split-collection Vehicle Routing Problem with Time Windows and Linear Weight-related Cost 37
  3.1 Introduction ........................................................... 37
  3.2 Problem Description, Properties and Formulation ............... 43
  3.3 Dantzig-Wolfe Decomposition ....................................... 48
    3.3.1 Master Problem .................................................. 48
    3.3.2 Pricing Subproblem .............................................. 50
  3.4 Column Generation .................................................. 52
    3.4.1 Extreme Collection Pattern ..................................... 53
    3.4.2 The Label-Setting Algorithm .................................... 56
    3.4.3 Implementation Details of Set Dominance Rule ............... 62
    3.4.4 Accelerating Strategies .......................................... 66
  3.5 Branch-and-Price-and-Cut Algorithm ................................ 76
    3.5.1 Valid Inequalities ............................................... 77
    3.5.2 Search strategy .................................................. 80
    3.5.3 Branching strategies ............................................. 81
3.6 Computational Experiments

3.6.1 Instances

3.6.2 Experimental Setup

3.6.3 Results on the SDVRPTW instances

3.6.4 Results on the SCVRPTWL instances

3.7 Conclusions

4 Vehicle Routing Problem with Stochastic Demands and Toll-by-Weight Scheme

4.1 Introduction

4.2 Problem description and properties

4.3 Approximation of the expected cost

4.4 Adaptive large neighborhood search

4.5 Computational experiments

4.6 Conclusions

5 Conclusions
List of Figures

2.1 Label Extension and Join .................................. 23
2.2 The sorted “LP Gap (%)” values associated with the BPC1 .......... 34
2.3 The sorted “LPC Gap (%)” values associated with the BPC2 .......... 35
2.4 The statistical results on the numbers of branch-and-bound nodes generated by BPC1 and BPC2 ................................. 35
3.1 The impact of vehicle weight on customer sequence ................. 38
3.2 An example of two patterns that have two customers in common 46
3.3 Graphic representation of the reduced cost function $G(r, q)$ ........... 57
3.4 The graphic representations of the functions $G^1(r^1, q)$ and $G^2(r^2, q)$ (note that the increasing parts of both functions have been replaced with zero slope pieces) .................................................. 60
3.5 (a) The graphic representations of functions $G^2(r^2, q)$, $G^3(r^3, q)$ and $G^4(r^4, q)$. (b) The graphic representation of function $G^i_{\min}(q)$ ......................................................... 61
3.6 (a) The graphic representations of functions $G^1(r^1, q)$, $G^2(r^2, q)$ and $G^3(r^3, q)$. (b) $G^i_{\min}(q)$ lies below $G^i(r^i, q)$ ......................................................... 61
3.7 An example of the dominance graph $G_i$ ................................. 63
3.8 The bidirectional search strategy .................................. 71
3.9 Graphic representation of the reduced cost function $G^b(r, Q - q)$ . 71
3.10 The graphic representation of $G(r, q)$ after the introduction of SMV inequalities .......................................................... 80
4.1 (a) Removal heuristics. (b) Insertion heuristics. . . . . . . . . . . 130
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Performance comparison between BPC1 and BPC2 on 30-vertex instances.</td>
<td>31</td>
</tr>
<tr>
<td>2.2</td>
<td>Performance comparison between BPC1 and BPC2 on 40-vertex instances.</td>
<td>32</td>
</tr>
<tr>
<td>2.3</td>
<td>Performance comparison between BPC1 and BPC2 on 50-vertex instances.</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Linear Relaxation Results on the SDVRPTW Instances.</td>
<td>85</td>
</tr>
<tr>
<td>3.2</td>
<td>Linear Relaxation Results on the SDVRPTW Instances in Group R2-100-100.</td>
<td>86</td>
</tr>
<tr>
<td>3.3</td>
<td>Integer Solution Results for the SDVRPTW Instances in Categories 1 and 2.</td>
<td>88</td>
</tr>
<tr>
<td>3.4</td>
<td>Linear Relaxation Results on the SCVRPTWL Instances.</td>
<td>89</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary of the Integer Solution Results to the SCVRPTWL Instances.</td>
<td>90</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary on the Number of Solved Instances.</td>
<td>91</td>
</tr>
<tr>
<td>3.7</td>
<td>Summary on the Number of Split Customers.</td>
<td>92</td>
</tr>
<tr>
<td>3.8</td>
<td>Optimal Integer Solutions for the SCVRPTWL instances (Part I).</td>
<td>93</td>
</tr>
<tr>
<td>3.9</td>
<td>Optimal Integer Solutions for the SCVRPTWL instances (Part II).</td>
<td>94</td>
</tr>
<tr>
<td>3.10</td>
<td>Optimal Integer Solutions for the SCVRPTWL instances (Part III).</td>
<td>95</td>
</tr>
<tr>
<td>4.1</td>
<td>Toll functions implemented by the seven Chinese Provinces.</td>
<td>125</td>
</tr>
</tbody>
</table>
4.2 Computational results of Class 1 instances. . . . . . . . . . . . . . 127
4.3 Computational results of Class 2 instances. . . . . . . . . . . . . . 128
4.4 Computational results of Class 3 instances. . . . . . . . . . . . . . 129
which arise from the industry in recent years. For example, after natural dis-
asters happen, it is very important for the rescue team (vehicles) to reach the
victims (customers) as soon as possible so as to reduce their suffering and life
losses. In this case, minimizing the arrival time of vehicles at customers is more
appropriate than minimizing the total travel distance of vehicles. Campbell et al.
[27] first addressed two routing problems for relief efforts which aimed to mini-
mize the maximum arrival time (minmax) and the average arrival time (minavg),
respectively. Another example where the classic objective fails to model the re-
ality arises in Chinese expressway where the amount of money paid by drivers to
the expressway companies depends on both the travel distance and the vehicle
weight, which is referred as the toll-by-weight scheme. Zhang et al. [131] first
introduced the toll-by-weight scheme into the context of VRP and proposed a
branch-and-bound algorithm to solve the resulted problem. However the algo-
rithm proposed by Zhang et al. [131] has a very serious drawback, that is, it can
only handle the problem with only one vehicle, which is very unrealistic.

Existing algorithms to solve the VRPs can be categorized into two classes:
exact algorithms and heuristics. Exact algorithms, including branch-and-bound
algorithm [31, 53], branch-and-cut algorithm [11, 14, 77, 91], branch-and-price
algorithm [37, 39, 43, 54] and etc., are a class of algorithms that are able to solve
the problem to optimality but may require huge computational effort. Most of
exact algorithms can only solve small or median-size instances. On the other
hand, heuristics, including tabu search [33, 58, 59, 124], simulated annealing
[5, 28, 126], genetic algorithm [12, 101, 102], adaptive large neighborhood search
[80, 106, 111] and etc, require less computational effort but cannot guarantee
to find the optimal solution. Many heuristics can solve practical-size instances
fast but fail to provide any information about the solution quality, like gaps to
optimality.
The vehicle routing problems with cumulative cost structure are generaliza-
tions of the classic vehicle routing problems where the travel cost of vehicle per
unit distance is a function of another quantity (e.g., number of customers served,
the weight of vehicle) which accumulates as the vehicle travels along the route.
Both the VRP which aims to minimize the total arrival time and the VRP with
the toll-by-weight scheme belong to the VRPs with cumulative cost structure. In
this thesis, we study three vehicle routing problems with cumulative cost struc-
ture, focusing on the design of effective exact algorithms and heuristics.

The first problem studied in this thesis is the multiple traveling repairmen
problem with distance constraints (MTRPD), which is an extension of the mul-
tiple traveling repairman problem (MTRP) by considering a limitation on the
total distance that a vehicle can travel. The maximum distance constraint usu-
ally stems from regulations on working hours for workers or arises in the home
delivery of perishable products. In the MTRPD, a fleet of vehicles is dispatched
to serve a set of customers. Each vehicle that starts from and ends at the depot
is not allowed to travel a distance longer than a predetermined limit and each
customer must be visited exactly once. The objective is to minimize the total
waiting time of all customers after the vehicles leave the depot, which is the same
as minimizing the total arrival time of vehicles at all the customers. The MTRP
can be viewed as a direct generalization of the classic traveling repairman problem
where only one vehicle is used. Applications of the TRP and MTRP can be found
in routing pizza deliverymen, routing automated guided vehicles through cells in
a flexible manufacturing system or scheduling machines to minimize mean flow
time for jobs [51]. To optimally solve the MTRPD, we have implemented two
branch-and-price-and-cut algorithms, which correspond to two types of label-
setting algorithms applied to the pricing subproblem. Experiments show that
the branch-and-price-and-cut algorithm that includes the bounded bi-directional
label-setting algorithm and space state relaxation outperforms the other one and is able to solve most of the test instances optimally.

Next we study the split-collection vehicle routing problem with time windows and linear weight-related cost (SCVRPTWL), which is a new vehicle routing problem variant that simultaneously involves time windows, split collection and linear weight-related cost. This problem consists of determining least-cost vehicle routes to serve a set of customers while respecting the restrictions of vehicle capacity and time windows. The travel cost of vehicle per unit distance is a linear function of the vehicle weight and the customer demand can be fulfilled by multiple vehicles. A real-life application of the vehicle routing problem with weight-related costs arises in Chinese expressway transportation system. At the end of 2013, over twenty five Chinese provinces have implemented toll-by-weight scheme in which expressway toll per unit distance is levied according to a monotonically increasing function $f(w)$ where $w$ is the weight of vehicle. In general, $f(w)$ is a piece-wise function with linear and quadratic parts. So under some moderate restrictions, our model can be applied in the toll-by-weight scheme. Moreover, we can also find applications from the transportation service providers who are concerned with fuel consumption and the environmental impact of greenhouse gas (GHG) emissions. The fuel expenditure accounts for a large portion of the overall transportation cost and thus greatly affects the profits of transportation service providers [127]. The fuel consumption rate is directly related to the vehicle weight; for example, for a vehicle of some type, its fully loaded status might consume more than twice as much diesel fuel as its empty status. In the last decade, the hazardous impacts of GHG, which is directly related to the consumption of fossil fuel, have received growing concerns from the public. Transport sector is one of the key sources of GHG emissions. As revealed by
U.S. Greenhouse Gas Inventory Report published in 2012, transportation activities account for 32% of U.S. CO₂ emissions from fossil fuel combustion in 2010. There is a clear tendency that transportation service providers will be forced to undertake the cost of their GHG emissions in the context of new regulations. The cost of fuel consumed or GHG emitted per unit distance by a vehicle with weight $w$ can be represented by a function $f(w)$. To solve this problem, we propose an exact branch-and-price-and-cut algorithm, where the pricing subproblem is a resource-constrained elementary least-cost path problem. We first prove that at least an optimal solution to the pricing subproblem is associated with an extreme collection pattern, and then design a tailored label-setting algorithm to solve it. Computational results show that our proposed algorithm can handle both the classic split delivery vehicle routing problem with time windows (SDVRPTW) and our problem effectively.

The third problem we investigate in this thesis is the vehicle routing problem with stochastic demands and the toll-by-weight scheme (VRPSD-TBW), which is a variant of the classic vehicle routing problem that considers the stochastic demands and the toll-by-weight scheme simultaneously. In China, different provinces have different tolling functions because of different considerations. In the VRPSD-TBW, we consider a set of tolling functions from 7 provinces, model this problem using the dynamic recourse model and derive the dynamic recursion to compute the expected cost of a route. The problem involves the design of a set of collection routes with minimal expected cost to cover a set of given customers. In order to solve the problem, we design an adaptive large neighborhood search (ALNS) heuristic based on different evaluation schemes which can approximately compute the expected cost of a route fast. We adopt the approximation evaluation schemes because the exact dynamic recursion is so time-consuming that the ALNS heuristic becomes extremely slow. To access the performance of our
ALNS heuristic and the approximation evaluation schemes, we propose a set of benchmark instances which are generated according to the real data from several Chinese provinces. Computational results show that the ALNS heuristic with the approximation evaluation schemes outperforms the one with the exact dynamic recursion and can handle the VRPSD-TBW effectively.

1.2 Contribution

This thesis examines three vehicle routing problems with cumulative cost structures, which are motivated by real-world applications. The models and solution approaches proposed in this thesis can be incorporated into the decision support system of the relative enterprises to provide better decision support for the managers. We list the contributions of this thesis as follows:

- Multiple Traveling Repairmen Problem with Distance Constraints
  1. Use two mixed integer programming models to formulate the problem: an arc-flow model and a set-covering model
  2. Propose two branch-and-price-and-cut algorithms, which correspond to two types of label-setting algorithms applied to the pricing subproblem
  3. Our computational results serve as benchmarks for future researchers on this problem

- Split-Collection Vehicle Routing Problem with Time Windows and Linear Weight-Related Cost
  1. Formulate the problem into an arc-flow model and a set-covering model, and analyze their properties
2. Develop an effective branch-and-price-and-cut algorithm to solve the problem to optimality

3. The branch-and-price-and-cut algorithm outperforms the existing exact algorithm for the split-delivery vehicle routing problem with time windows

- Vehicle Routing Problem with Stochastic Demands and the Toll-by-Weight Scheme
  1. Incorporate the toll-by-weight scheme into the vehicle routing problem with stochastic demands
  2. Propose an effective adaptive large neighborhood search based on different approximation evaluation schemes
  3. Contribute a set of benchmark instances generated according to real-world data

1.3 Thesis Organization

This thesis is organized as follows. We study the Multiple Traveling Repairmen Problem with Distance Constraints in Chapter 2. In Chapter 3, we investigate the Split-Collection Vehicle Routing Problem with Time Windows and Linear Weight-Related Cost. We study the Vehicle Routing Problem with Stochastic Demands and the Toll-by-Weight Scheme in Chapter 4. In Chapter 5, we conclude the thesis, summarizing the academic contributions and suggesting some possible future research.
Chapter 2

Branch-and-Price-and-Cut for the Multiple Traveling Repairman Problem with Distance Constraints

2.1 Introduction

The traveling repairman problem (TRP) has been extensively studied by a large number of researchers (e.g., Afrati et al. [2], García et al. [55], Salehipour et al. [114]); this problem is also termed the minimum latency problem [25], the traveling deliveryman problem [93], and the cumulative traveling salesman problem [24]. The TRP is defined on a complete graph $G = (V, E)$, where $V = \{0, 1, \ldots, n, n + 1\}$ is the vertex set and $E = \{(i, j) : i, j \in V, i \neq j, i \neq n + 1, j \neq 0\}$ is the edge set. Vertices 0 and $n + 1$ represent the exit from and the entrance to the depot, respectively. We denote the vertices representing the set of $n$ customers by $V_C = \{1, \ldots, n\}$. The repairman (henceforth referred to as vehicle) is assumed to travel at a constant speed. Each edge $(i, j)$ has a non-negative length $d_{i,j}$ and requires a non-negative traversing time $t_{i,j}$, which is
symmetric, i.e., \( t_{i,j} = t_{j,i} \), and satisfies the triangle inequality rule. The objective of the TRP is to find a Hamiltonian tour on \( G \), starting from vertex 0 and ending at vertex \( n + 1 \), which minimizes \( \sum_{i \in V_c} l_i \), where \( l_i \) denotes the waiting time of customer \( i \) after the vehicle leaves vertex 0. A direct generalization of the TRP is the multiple traveling repairman problem (MTRP) that considers \( K \) identical vehicles \([48]\). Applications of the TRP and MTRP can be found in routing pizza deliverymen, routing automated guided vehicles through cells in a flexible manufacturing system or scheduling machines to minimize mean flow time for jobs \([51]\).

This chapter studies an extension of the MTRP by involving a distance constraint that the route length (or duration) of each vehicle cannot exceed a predetermined limit \( L \). This type of constraint usually stems from regulations on working hours for workers or arises in the home delivery of perishable products. We call the resulting problem the \textit{multiple traveling repairman problem with distance constraints} (MTRPD) whose objective is to find \( K \) routes such that each vertex is visited exactly once, the distance constraint is respected and the total waiting time of all customers is minimized. Examples of other vehicle routing models that incorporate the distance constraint can be found in Laporte et al. \([83]\), Li et al. \([86]\), Erera et al. \([47]\).

The MTRP can be viewed as a variant of the \textit{multiple traveling salesman problem} (\textit{m-TSP}) \([123, 17]\). Although many researchers have studied the TRP, the literature on the MTRP is surprisingly limited. The only prior study we can find in existing literature is Fakcharoenphol et al. \([48]\). They presented a polynomial-time \( 8.497 \gamma \)-approximation algorithm for the MTRP, where \( \gamma \) denotes the best polynomial-time approximation factor possible for the \( k \) minimum spanning tree (\( k \)-MST) problem \([10]\).

Apart from the MTRPD, there exist several other extensions of the MTRP
in the literature. Bennett and Gazis [19] introduced a school bus routing problem (SBRP) in which a fleet of school buses is dispatched to take pupils from pick-up points to school. Each bus has a fixed capacity and each pick-up point has a given demand, represented by the number of pupils. The objective of this problem is to minimize the weighted sum of the total bus travel time and the total pupil travel time. Li and Fu [87] described a case study of routing school bus for Hong Kong kindergartens. They formulated the problem as a multi-objective combinatorial optimization problem with four types of objectives, which are prioritized in the following order: (1) minimize the total number of buses required; (2) minimize the total travel time spent by all pupils; (3) minimize the total bus travel time; and (4) balance the loads and travel times among all buses. For an overview of the SBRP, we refer the reader to Park and Kim [98].

In the SBRP, it is obvious that the demand at each pick-up point must be integer. When the vertex demand is allowed to be a real number, the resulting problem is called the cumulative vehicle routing problem (CumVRP) [74, 94, 106]. The CumVRP is the same as the classical capacitated vehicle routing problem (CVRP) [125] except that the cost of traversing an edge is the product of length and flow of the edge. It has been shown that the MTRP is a special case of the delivery formulation of the CumVRP; we refer the reader to Kara et al. [74] for details of the proof. The CumVRP can be further regarded as a special case of the weighted vehicle routing problem (WVRP) proposed by Zhang et al. [129]. The total cost to be minimized in the WVRP consists of three components: (1) the fixed cost of dispatching a vehicle; (2) the cost per unit travel distance; and (3) the constant surcharge per unit weight per unit distance. Later, the WVRP was extended to include multiple depots by Zhang et al. [130].

When the surcharge per unit weight per unit distance is a function of the vehicle weight, the WVRP is generalized to the vehicle routing problem with
toll-by-weight scheme (VRPTBW) [119, 131]. To date, over twenty five Chinese provinces have implemented the toll-by-weight schemes, all of which are monotonically increasing functions of the vehicle weight. Denoting by the decision variable \( w_{i,j} \) the weight of the vehicle traversing edge \((i, j)\) and assuming the surcharge per unit distance is calculated based on a toll function \( f(w_{i,j}) \), the objective of the VRPTBW is to minimize \( \sum_{i \in V} \sum_{j \in V} d_{i,j} f(w_{i,j}) \). Consider the case where the toll function has the following form:

\[
  f(w_{i,j}) = \begin{cases} 
    0 & \text{if } w_{i,j} = 0 \\
    \alpha w_{i,j} + \beta & \text{if } w_{i,j} > 0 
  \end{cases}
\]

where \( \alpha \) and \( \beta \) are non-negative parameters. If \( \alpha = 0 \) and \( \beta = 1 \), then VRPTBW reduces to the classical CVRP problem. If \( \alpha = 1 \) and \( \beta = 0 \), then VRPTBW reduces to the CumVRP. If \( \alpha > 0 \) and \( \beta = 0 \), then \( \sum_{i \in V} \sum_{j \in V} d_{i,j} f(w_{i,j}) \) can be written as \( \sum_{i \in V} \sum_{j \in V} \alpha d_{i,j} w_{i,j} \), which is actually the third cost component of the WVRP [129].

The workover rig routing problem (WRRP) introduced by Aloise et al. [6] is another variant of the MTRP. In the WRRP, a set of onshore oil wells needs maintenance service from a fleet of heterogeneous workover rigs. For each well, its production loss equals the product of the production loss rate and the time at which its required service is completed. The objective of this problem is to find a route for each workover rig such that the total production loss of the wells over a finite horizon is minimized. In recent years, the WRRP has also been studied by several other researchers, such as Pacheco et al. [96], Ribeiro et al. [108, 107].

After reviewing prior studies with regard to the MTRP and its variants, we find that almost all relative articles except Ribeiro et al. [107] proposed near-optimal algorithms, such as approximation algorithm [48], heuristics [19, 87], scatter search algorithms [129, 130], simulated annealing algorithm [119] and
variable neighborhood search \cite{6}. In Ribeiro et al. \cite{107}, the authors proposed a branch-and-price-and-cut algorithm for the WRRP, where their column generation pricing subproblem was solved by a mono-directional label-setting algorithm.

In this chapter, we provide an exact branch-and-price-and-cut algorithm for the MTRPD. The pricing subproblem of the MTRPD is called the resource-constrained elementary shortest path problem with cumulative costs. Although label-setting algorithms have been successfully applied to similar pricing subproblems in several previous articles, e.g., Ribeiro et al. \cite{107} and Ioachim et al. \cite{69}, our work is the first attempt to develop bounded bidirectional label-setting algorithm for solving it. To the best of our knowledge, branch-and-price-and-cut method is the most successful exact algorithm for the vehicle routing models \cite{54,13}, such as the vehicle routing problem with time windows (VRPTW) \cite{42,72}, the split delivery VRPTW \cite{39,8}, the capacitated location-routing problem \cite{32}, the heterogeneous fleet vehicle routing problem \cite{99} and the pickup and delivery problem with time windows \cite{112}.

The remainder of the chapter is structured as follows. Section 2.2 presents an arc-flow formulation and a set-covering formulation for the MTRPD. This is followed in Section 2.3 with a description of column generation, consisting of the pricing subproblem, the label-setting algorithm for solving the pricing subproblem and three acceleration strategies. Subsequently, we present other main components of the branch-and-price-and-cut algorithm in Section 2.4. Our experimental results are given in Section 2.5 and we conclude our chapter in Section 2.6 with some closing remarks.

### 2.2 Mathematical formulation

The arc-flow formulation of the MTRPD uses two types of decision variables: a binary decision variable $x_{i,j,k}$ that equals 1 if vehicle $k$ directly travels from vertex
i to vertex j, and 0 otherwise; and a non-negative variable \( y_{i,k} \) that represents the time at which vehicle \( k \) arrives at vertex \( i \). We denote by \( V^+(i) = \{ j \in V | (i,j) \in E \} \) and \( V^-(i) = \{ j \in V | (j,i) \in E \} \) the immediate successors and predecessors of vertex \( i \) in \( G \). Letting \( F \) be the set of \( K \) vehicles and \( M \) be a sufficiently large positive number, the arc-flow formulation is given as:

\[
\begin{align*}
    z &= \min \sum_{k \in F} \sum_{i \in V_C} y_{i,k} \\
    \text{s.t.} & \quad \sum_{k \in F} \sum_{j \in V^+(i)} x_{i,j,k} = 1, \forall i \in V_C \\
    & \quad \sum_{j \in V^+(0)} x_{0,j,k} = 1, \forall k \in F \\
    & \quad \sum_{j \in V^+(i)} x_{i,j,k} = \sum_{j \in V^-(i)} x_{j,i,k}, \forall k \in F, i \in V_C \\
    & \quad \sum_{i \in V^-(n+1)} x_{i,n+1,k} = 1, \forall k \in F \\
    & \quad y_{j,k} \geq y_{i,k} + t_{i,j} + M(x_{i,j,k} - 1), \forall k \in F, (i,j) \in E \\
    & \quad \sum_{k \in F} \sum_{(i,j) \in E \setminus S} x_{i,j,k} \geq 1, \forall S \subset V_C \\
    & \quad 0 \leq y_{i,k} \leq L, \forall k \in F, i \in V \\
    & \quad x_{i,j,k} \in \{0,1\}, \forall k \in F, (i,j) \in E
\end{align*}
\]

The objective function (2.1) aims at minimizing the total waiting time of all customers. Constraints (2.2) ensure that each customer must be visited exactly once. Constraints (2.3) and (2.5) guarantee that each vehicle starts from vertex 0 and ends at vertex \( n + 1 \). Constraints (2.4) are the flow conservation constraints. The relationship between the arrival times at two consecutive vertices visited by the same vehicle should satisfy Constraints (2.6) and each arrival time is restricted within interval \([0,L]\) by Constraints (2.8). Constraints (2.7) are subtour elimination (SE) cuts that are redundant but can tighten the linear relaxation of the formulation. If we remove Constraints (2.7), the remaining formulation can
be directly handled by some commercial integer programming solvers, e.g., ILOG CPLEX.

After conducting some preliminary experiments, we find that the size of the instances optimally solved by CPLEX is quite limited. To achieve optimal solutions for instances of practical size, we reformulate the problem as a set-covering model through Dantzig-Wolfe decomposition and develop an exact branch-and-price-and-cut algorithm to solve it. Let $\Omega$ be the set of all feasible routes for the vehicles. A route $r \in \Omega$ can be written as $r = (v_0, v_1, \ldots, v_{|r|}, v_{|r|+1})$, where $|r|$ is the number of vertices covered by route $r$, $v_0 = 0$ and $v_{|r|+1} = n + 1$. The cost of route $r$ is known in advance by:

$$c_r = \sum_{i=1}^{|r|} \sum_{j=0}^{j=i-1} t_{j,j+1}$$

We denote by parameters $a_{i,r}$ and $b_{i,j,r}$ the number of times route $r$ visits customer $i \in V_C$ and the number of times route $r$ traverses edge $(i, j)$, respectively. In our problem, each customer can be visited exactly once, so parameters $a_{i,r}$ and $b_{i,j,r}$ must be binary. If the routes containing cycles are allowed, these parameters can take integers greater than one. Moreover, for each route $r \in \Omega$ we define a binary variable $\theta_r$ which takes one if this route is selected in the solution, and zero otherwise. Obviously, each $\theta_r$ corresponds to a column $a_r = (a_{1,r}, \ldots, a_{n,r})$. For the sake of brevity, we further define $\theta$ to be a vector variable that contains all $\theta_r$. With the above notations, the set-covering formulation is as follows:

$$z = \min \sum_{r \in \Omega} c_r \theta_r$$

s.t. $\sum_{r \in \Omega} a_{i,r} \theta_r \geq 1, \ \forall \ i \in V_C$ \hfill (2.10)

$$\sum_{r \in \Omega} \theta_r \leq K$$ \hfill (2.11)

$$\theta_r \in \{0, 1\}, \ \forall \ r \in \Omega$$ \hfill (2.12)
The objective function (2.9) minimizes the total cost of the selected routes. Constraints (2.10) ensure that every customer is visited at least once, which enlarges the feasible region of the original problem by allowing each customer to be visited more than once. However, the optimal solutions of the modified problem must be the same as those of the original problem if the triangle inequality rule is satisfied. In other words, the optimal solutions of the set-covering formulation would not visit any customer more than once. Since only \( K \) vehicles are available, Constraint (2.11) applies. Like Constraints (2.7) in the arc-flow formulation, we can also incorporate the SE cuts to tighten the set-covering formulation, which has the following form:

\[
\sum_{(i,j) \in E: i \notin V \setminus S, j \in S} \sum_{r \in \Omega} b_{i,j,r} \theta_r \geq 1, \forall \ S \subset V_C
\]  

(2.13)

In practice, even for a small-size instance, model (2.9) – (2.13) contains a huge number of variables (i.e., columns) and SE cuts, which cannot be written out explicitly. Hence, this model cannot be directly handled by CPLEX. Our proposed branch-and-price-and-cut algorithm is not required to enumerate all columns but generate new columns as needed. Moreover, we relax all SE cuts at the beginning of the algorithm, and add some of them into the model over the course of the branch-and-bound search process. The branch-and-price-and-cut algorithm uses a branch-and-bound framework, where at each search tree node column generation is applied to compute a lower bound and SE cuts are dynamically added to further improve the lower bound.

In the remaining chapter, we distinguish between the terms node and vertex, which are usually considered the same and are used interchangeably in standard graph terminology; we specify that node refers to the branch-and-bound search tree node, while vertex refers to the vertex in graph \( G \).
2.3 Column generation

Column generation [41] is applied to solve the linear relaxation of the model (2.9) – (2.12) augmented by appropriate branching decisions and SE cuts, which is called the *linear master problem* (LMP). The optimal solution value of the LMP is a lower bound of its associated branch-and-bound node. Column generation cannot directly solve the LMP due to its inability to enumerate all columns. Instead, it is an iterative procedure that alternates between solving a *restricted linear master problem* (RLMP) and a *pricing subproblem*. The RLMP is the LMP restricted to a subset \( \overline{\Omega} \subseteq \Omega \) of columns, which can be optimally solved by simplex method. The goal of solving the pricing subproblem is to identify the columns in \( \Omega \setminus \overline{\Omega} \) that have negative reduced costs with respect to the dual optimal solution of the current RLMP. If no such column is found, we terminate the column generation procedure with an optimal solution of the LMP, which equals the optimal solution of the current RLMP. Otherwise, we incorporate the identified negative reduced cost columns into the current RLMP and restart the column generation iteration.

In this section, we focus on describing the column generation procedure applied to the LMP at the root branch-and-bound node, which does not involve branching constraints. We first present the pricing subproblem and then describe a basic (mono-directional search) label-setting algorithm to optimally solve it. Subsequently, we detail four strategies to accelerate the column generation procedure, which are *tabu search column generator*, *bounded bidirectional search*, *state space relaxation* and *decremental state-space relaxation*. The LMPs at other branch-and-bound nodes can be solved using this column generation procedure with minor modifications, which is elaborated in Section 2.4.4.
2.3.1 The pricing subproblem

At each branch-and-bound node, the column generation procedure and the cut separation algorithm are performed alternatively. The cut separation algorithm is invoked immediately after the column generation procedure. Once new SE cuts are added into the RLMP, the column generation procedure is restarted to solve the modified RLMP. We assume that at the root node some columns and SE cuts have already been added into the RLMP, which therefore has the following form:

\[
\min_{r \in \Omega} \sum_{r \in \Omega} c_r \theta_r \quad (2.14)
\]

subject to

\[
\sum_{r \in \Omega} a_{i,r} \theta_r \geq 1, \quad \forall \ i \in V_C \quad (2.15)
\]

\[
\sum_{r \in \Omega} \theta_r \leq K \quad (2.16)
\]

\[
\sum_{(i,j) \in E : i \in V \setminus S, j \in S} \sum_{r \in \Omega} b_{i,j,r} \theta_r \geq 1, \quad \forall \ S \in \Gamma \quad (2.17)
\]

\[
0 \leq \theta_r \leq 1, \quad \forall \ r \in \Omega \quad (2.18)
\]

where each element in set \(\Gamma\) is a subset of \(V_C\), representing an individual SE cut.

We check whether the solution of the LMP reaches optimality by solving the pricing subproblem. Below we will use \(\pi = (\pi_1, \ldots, \pi_n), \mu\) and \(\lambda = (\lambda_{S_1}, \ldots, \lambda_{S_{|\Gamma|}})\) as the dual variables associated with Constraints \((2.15), (2.16)\) and \((2.17)\), respectively. Assuming that \((\hat{\pi}, \hat{\mu}, \hat{\lambda})\) is the dual optimal solution of the current RLMP, the reduced cost \(\bar{c}_r\) associated with route \(r \in \Omega\) is calculated by:

\[
\bar{c}_r = c_r - \left( \sum_{i \in V_C} \hat{\pi}_i a_{i,r} - \hat{\mu} + \sum_{S \in \Gamma} \hat{\lambda}_S \sum_{(i,j) \in E : i \in V \setminus S, j \in S} b_{i,j,r} \right) \quad (2.19)
\]

Based on the above expression, we can formulate the pricing subproblem as
follows:

\[
\begin{align*}
z^{PS} &= \min \sum_{i \in V_C} \left\{ y_i - \tilde{\pi}_i \sum_{j \in V^+(i)} x_{i,j} \right\} - \sum_{S \in \Gamma} \sum_{(i,j) \in E : i \in V \setminus S, j \in S} \lambda_S x_{i,j} + \mu \quad (2.20) \\
\text{s.t.} & \sum_{j \in V^+(0)} x_{0,j} = 1 \quad (2.21) \\
& \sum_{i \in V^{-(n+1)}} x_{i,n+1} = 1 \quad (2.22) \\
& \sum_{j \in V^+(i)} x_{i,j} = \sum_{j \in V^-(i)} x_{j,i} \leq 1, \forall i \in V_C \quad (2.23) \\
& y_j \geq y_i + t_{i,j} + (x_{i,j} - 1)M, \forall (i, j) \in E \quad (2.24) \\
& 0 \leq y_i \leq L, \forall i \in V \quad (2.25) \\
& x_{i,j} \in \{0, 1\}, \forall (i, j) \in E \quad (2.26)
\end{align*}
\]

where variable \( y_i \) denotes the arrival time at vertex \( i \) and variable \( x_{i,j} \) represents the number of times edge \((i, j)\) is traversed. The objective function (2.20) aims to achieve the minimal reduced cost of all feasible routes. The reduced cost consists of three parts: (1) the arrival time \( y_i \) of each visited customer, (2) the prize \(-\tilde{\pi}_i\) at each visited customer; and (3) the prizes associated with some edges traversed by the route. Constraints (2.21) and (2.22) require that the route must start from vertex 0 and end at vertex \( n+1 \). Constraints (2.23) ensure that each customer can be visited at most once. Constraints (2.24) define the relationship between the arrival times of two consecutively visited vertices. The length of the route is restricted to be less than or equal to \( L \) by Constraints (2.25). Solving the pricing subproblem is essentially equivalent to enumerating all feasible routes in \( \Omega \). This pricing subproblem has been shown to be \( NP \)-complete in Appendix A, implying that optimally solving its instances is computationally expensive.

In many articles that proposed branch-and-price algorithms for solving the vehicle routing models, the pricing subproblems are usually the elementary shortest path problem with resource constraints (ESPPRC) \[49\]. A path is called elementary if each vertex is visited at most once. Examples of the ESPPRC can be
found in Desrochers et al. [43], Gutiérrez-Jarpa et al. [62] and Azi et al. [11]. Our pricing subproblem is different from the traditional ESPPRC since it considers a cumulative cost $y_i$ at each visited customer $i$. It is a special case of the pricing subproblem of the WRRP Ribeiro et al. [107]. Moreover, its version that allows the routes with cycles can be regarded as a special case of the shortest path problem with time windows and time costs (SPPTWTC) introduced by Ioachim et al. [69]. Both the SPPTWTC and the pricing subproblem of the WRRP have been solved by mono-directional label-setting algorithms. However, we do not find previous articles that applied the label-setting algorithms armed with bidirectional search strategy to solve them or similar problems.

In Ribeiro et al. [107], the authors claimed that the bidirectional search strategy cannot be applied to the pricing subproblem of the WRRP. We cannot make a conclusion on whether there exists a bidirectional label-setting algorithm for their pricing subproblem. However, after carefully analyzing the structure of our pricing subproblem, we find that it can still be solved by a tailored bounded bidirectional label-setting algorithm, which will be detailed in this section.

### 2.3.2 The label-setting algorithm

We optimally solve the pricing subproblem using a label-setting algorithm, which has been applied to the ESPPRC [49, 110] and the shortest path problem with resource constraints (SPPRC) [70]. The aim of solving the pricing subproblem is to identify the negative reduced cost columns. In our label-setting algorithm, a multi-dimensional label $S_i = (C_i, L_i, V_{i,1}, \ldots, V_{i,n})$ is defined to represent a state associated with a partial route from vertex 0 to vertex $i$, where:

- $C_i$ is the reduced cost of this partial route;
- $L_i$ is the amount of distance the vehicle has traveled;
• $V_{i,k}$ (1 ≤ $k$ ≤ $n$) is a dummy resource indicating whether customer $k$ has already been visited.

At vertex 0, the values of $L_0$ and all $V_{0,k}$ (1 ≤ $k$ ≤ $n$) are initialized to zero and the value of $C_0$ is set to $\hat{\mu}$. The value of $V_{i,k}$ would be set to one when vertex $k$ either is covered by the route or cannot be visited in any extension due to the distance constraint. Each vertex may have multiple labels and the optimal solution of the pricing subproblem can be achieved by identifying the minimum-cost label at vertex $n + 1$. Note that labels do not contain any information regarding the order in which the vertices have been visited.

A label $S_i = (C_i, L_i, V_{i,1}, \ldots, V_{i,n})$ can be extended to vertex $j \in V^+(i)$, generating a new label $S_j = (C_j, L_j, V_{j,1}, \ldots, V_{j,n})$. We first identify all vertices $j$ that cannot be reached from $S_i$, namely $L_i + t_{i,j} > L$, and accordingly set $V_{i,j} = 1$. For the remaining vertices in $V^+(i)$, the labels $S_j$ are created according to the following extension functions:

$$L_j = L_i + t_{i,j}$$

$$C_j = C_i + L_j - \hat{\pi}_j - \sum_{S \in \Gamma: j \in V \setminus S, j \in S} \hat{\lambda}_S$$

$$V_{j,k} = \begin{cases} 
V_{i,k} + 1, & \text{if } k = j \\
V_{i,k}, & \text{if } k \neq j 
\end{cases}$$

where we set $\hat{\pi}_0 = \hat{\pi}_{n+1} = 0$ since each route must start from vertex 0 and end at vertex $n + 1$. In the course of the label-setting algorithm, we cyclically examine all vertices, at each of which all labels that do not have successors would be extended.

Extending a label at vertex $i$ may create as many new labels as the number of its successors. Undoubtedly, the number of labels would increase exponentially with the extension of the labels. To avoid enumerating all labels, a dominance rule is employed in our label-setting algorithm to identify and eliminate the dominated labels. Given two labels $S_i$ and $\bar{S}_i$, $\bar{S}_i$ is dominated by $S_i$ and thus can be safely
discarded if $C_i \leq \bar{C}_i$, $L_i \leq \bar{L}_i$, $V_{i,k} \leq \bar{V}_{i,k}$ for all $1 \leq k \leq n$, and at least one of the above inequalities is strict. We arbitrarily discard one of two identical labels.

2.3.3 Accelerating strategies

We have implemented the following four techniques to speed up the column generation procedure.

2.3.3.1 Tabu search column generator

The tabu search algorithm [60] we develop for solving the pricing subproblem is similar to the one proposed by Desaulniers et al. [42]. We run the tabu search algorithm multiple times with different initial routes, derived from the basic variables in the optimal solution of the current RLMP. Three local search operators are used: (1) inserting a vertex into the current route, (2) removing a vertex from the current route, and (3) swapping two vertices, one visited by the route and the other unvisited. Each operator corresponds to a neighborhood of the current route and only the neighbors satisfying the distance constraint are retained. The tabu search algorithm keeps a tabu list to prevent the search process from being trapped in local optima. Moves specified by the tabu list are not allowed for $\xi$ iterations, where $\xi$ is a controlling parameter.

The tabu search process iteratively proceeds to the best allowable neighbor (i.e., the neighbor with the least reduced cost) of the current route until the number of iterations exceeds $maxIter$. We terminate the tabu search algorithm when either all initial solutions have been used up or $maxCol$ routes with negative reduced cost have been obtained. The tabu search stage is able to rapidly identify high quality solutions to the pricing subproblem, whereas we cannot prove whether these solutions are optimal or not. As a result, when the tabu search algorithm fails to identify a negative reduced cost route, we still need to invoke
the exact label-setting algorithm.

2.3.3.2 Bounded bidirectional search

The bounded bidirectional search strategy is a useful technique for accelerating the label-setting algorithm \[109, 110\]; the resulting algorithm is called the *bounded bidirectional label-setting algorithm (BBLS)*. In the BBLS algorithm, partial routes are originated from both vertices 0 and \( n + 1 \), and accordingly associated labels are called *forward labels* \( S_{fw}^i \) and *backward labels* \( S_{bw}^j \). Moreover, each forward (resp. backward) partial route can be extended only if its length is less than \( L/2 \), i.e., \( L_{fw}^i < L/2 \) (resp. \( L_{bw}^j < L/2 \)). At vertex 0, we set \( C_{fw}^0 = \hat{\mu}/2 \), \( L_{fw}^0 = 0 \) and \( V_{0,k}^{fw} = 0 \) for all \( 1 \leq k \leq n \). The extension functions and dominance rule on forward labels \( S_{fw}^i = (C_{fw}^i, L_{fw}^i, V_{i,1}^{fw}, \ldots, V_{i,n}^{fw}, V_{i,n+1}^{fw}) \) have been described in Section 2.3.2. Hereafter, we describe the extension functions and dominance rule for backward labels.

We use a label \( S_{bw}^j = (C_{bw}^j, L_{bw}^j, D_{bw}^j, V_{j,1}^{bw}, \ldots, V_{j,n}^{bw}) \) to represent a state in backward extension. The \( V_{j,k}^{bw} \) associated with all unreachable vertices \( k \) from vertex \( j \) are set to 1. Compared with the forward label, the backward label has an extra element \( D_{bw}^j \) that denotes the number of customers the backward partial route has visited upon arrival at vertex \( j \). The value of \( C_{bw}^n \) is set to \( \hat{\mu}/2 \) and the values of \( L_{n+1}^{bw}, D_{n+1}^{bw} \) and \( V_{n+1,k}^{bw} \) for all \( 1 \leq k \leq n \) are set to zero. Figure 2.1 pictorially shows the forward and backward extensions. When the backward label is extended from vertex \( j \) to vertex \( j + 1 \), the waiting time \( l_j \) of vertex \( j \) equals the waiting time \( l_{j+1} \) of vertex \( j + 1 \) plus \( t_{j+1,j} \). However, we are not able to know the actual waiting time of each vertex on the backward partial route unless the complete route is achieved. Consequently, we can only say that the waiting time of vertex \( j \) is at least \( t_{j+1,j} \). When the label is further extended to vertex \( j + 2 \), the waiting time of vertex \( j \) would become \( l_j + t_{j+2,j+1} + t_{j+1,j} \), and the waiting time of vertex \( j + 1 \) is \( l_{j+2} + t_{j+2,j+1} + t_{j+1,j+1} \). Thus, the total waiting time should be at least \( 2t_{j+2,j+1} + t_{j+1,j} \). Based on the above observations, when \( L_{bw}^j + t_{i,j} > L \), we
derive the functions for the backward extension from vertex $j$ to vertex $i$ as:

$$C_{bw}^i = C_{bw}^j + D_{bw} t_{i,j} - \sum_{S \in \Gamma \setminus \{j\} : i \in S, j \in S} \hat{\pi}_S$$

$$L_{bw}^i = L_{bw}^j + t_{i,j}$$

$$D_{bw}^i = D_{bw}^j + 1$$

$$V_{bw}^i = \begin{cases} V_{bw}^j + 1, & \text{if } k = i \\ V_{bw}^j, & \text{if } k \neq i \end{cases}$$

The cost $C_{bw}^j$ of the backward partial route counts in the already known minimum waiting times rather than the actual waiting times of all visited customers. Label $S_{bw}^j$ dominates label $\bar{S}_{bw}^j$ if $C_{bw}^j \leq \bar{C}_{bw}^j$, $L_{bw}^j \leq \bar{L}_{bw}^j$, $D_{bw}^j \leq \bar{D}_{bw}^j$, $V_{bw}^j \leq \bar{V}_{bw}^j$ for all $1 \leq k \leq n$ and at least one of these inequalities is strict. We also arbitrarily discard one of two identical backward labels.

A forward label $S_{fw}^p = (C_{fw}^p, L_{fw}^p, V_{fw}^1, \ldots, V_{fw}^p)$ and a backward label $S_{bw}^q = (C_{bw}^q, L_{bw}^q, D_{bw}^q, V_{bw}^1, \ldots, V_{bw}^q)$ can be joined together to form a complete feasible route if both of the following conditions hold:

$$L_{fw}^p + L_{bw}^q + t_{p,q} \leq L$$

$$V_{fw}^p + V_{bw}^q \leq 1, \quad \forall 1 \leq k \leq n$$

When a backward label $S_{bw}^q$ is concatenated to a forward label $S_{fw}^p$, the waiting time at each of $D_{bw}^q$ customers visited by the backward partial route must increase
by $L_q^w + t_{p,q}$. Hence, the cost of the resulting complete route is:

$$C_p^{fw} + C_q^{bw} + D_q^{bw} \times (L_q^w + t_{p,q}) - \sum_{S \in \Gamma : p \in V \setminus S, q \in S} \lambda_S$$

The minimum cost among all complete routes is the optimal solution value of the pricing subproblem. Usually, the BBLS algorithm can find a number of negative reduced cost columns and therefore we specify that at most $maxCol$ columns can be added into $\bar{\Omega}$.

### 2.3.3.3 State space relaxation

Another common way to accelerate the label-setting algorithm is to relax the elementarity requirements of all vertices, which was first introduced by Christofides et al. [30] under the name state space relaxation. If vertices are allowed to be visited more than once, optimally solving the pricing subproblem can be done in pseudo-polynomial time and therefore requires less computation time. However, with state space relaxation the label-setting algorithm may generate some columns with cycles, which are infeasible to the original problem. When $\Omega$ is re-defined to include routes with cycles, the resultant LMP would have an optimal objective value not greater than that of the original LMP, i.e., the lower bound at each branch-and-bound node may be weakened. Allowing routes with cycles may increase the computational burdens on the branch-and-price-and-cut algorithm because the weaker lower bound reduces the efficiency of fathoming nodes, increasing the number of the branch-and-bound tree nodes. Briefly speaking, there is a tradeoff between the computation speed of the pricing subproblem and the scale of the branch-and-bound tree.

In the BBLS algorithm with state space relaxation, forward and backward labels are defined as $S_i^{fw} = (C_i^{fw}, L_i^{fw})$ and $S_j^{bw} = (C_j^{bw}, L_j^{bw}, D_j^{bw})$, respectively. The manners employed to extend labels, judge dominance relationships and join
labels are the same as those described in Section 2.3.3.2 except that all $V_{j,k}$ $(1 \leq k \leq n)$ are not considered any more.

### 2.3.3.4 Decremental state-space relaxation

The decremental search state-space relaxation was introduced independently by Boland et al. [26] and Righini and Salani [110]. It starts from solving the pricing subproblem with the elementary requirements of all customers being relaxed, i.e., each customer can be visited more than once in a route. If the computed least-cost path is nonelementary, the customers that are visited more than once are required to be elementary and the pricing subproblem is solved again. This process is repeated until an elementary least-cost route is found. Our implementation of the decremental state-space relaxation technique is the same as the one described in Righini and Salani [110]. This acceleration technique has also been employed in the branch-and-price algorithms for solving several other vehicle routing models, such as the VRP with discrete split deliveries and time windows [113], the VRP with deliveries, selective pickups and time windows [62] and the multi-depot VRPTW [22].

### 2.4 Branch-and-price-and-cut algorithm

In this section, we first describe an approach for generating a set of initial columns and an upper bound for the problem. Next, we present a separation algorithm to identify the violated SE cuts. This is followed by search and branching strategies that guide the exploration of the branch-and-bound tree.
2.4.1 Initial columns and upper bound

We obtain a set of initial columns and an upper bound for the problem from a feasible solution produced by a *cheapest insertion* (CI) heuristic. The upper bound is updated over the course of the branch-and-bound search process.

A state in the CI heuristic is a feasible partial solution consisting of unvisited vertices and a set of feasible routes, one of which is designated as the current route. The CI heuristic begins with an empty route as the current route. We repeatedly insert into the current route the best unvisited vertex in terms of increased cost. When no further vertex can be inserted without violating the distance constraint, we continue with another empty route. The above process is repeated until all vertices are inserted.

2.4.2 Separation algorithm

To identify the violated SE cuts, we apply the polynomial-time separation algorithm introduced by Kohl et al. [75]; this algorithm is reviewed as follows. Assuming that \( \hat{\theta} \) is an optimal solution of the LMP, the flow on each edge \((i, j)\) can be calculated according to \( \hat{x}_{i,j} = \sum_{r \in \bar{\Omega}} b_{i,j,r} \hat{\theta}_r \). To check whether there exist some violated SE cuts, we consider the graph \( G \) and impose a capacity \( \hat{x}_{i,j} \) on edge \((i, j)\). For each vertex \( i \in V_C \), we solve a minimum cut problem with source vertex 0 and sink vertex \( i \) on graph \( G \). Let \( S_{\text{min}} \) denote the vertex set that contains vertex \( i \) and is separated by the minimum cut. If the total capacity of the minimum cut is less than one, namely \( \sum_{(i,j) \in E : i \in V \setminus S_{\text{min}}, j \in S_{\text{min}}} \hat{x}_{i,j} < 1 \), then the SE cut corresponding to \( S_{\text{min}} \) is violated, which is thereupon added into the RLMP (i.e., add \( S_{\text{min}} \) into \( \Gamma \)); otherwise, all SE cuts are satisfied. Finding a minimum cut in a directed graph is equivalent to solving a maximum flow problem [3], which can be done in polynomial time.
2.4.3 Search strategy

The branch-and-bound tree is explored according to a best-first policy; specifically, the “best” unexamined tree node is the one with the smallest lower bound, and would be given the highest priority. We have tested the depth-first policy in some preliminary experiments and obtained inferior results in terms of the number of the optimally solved instances within the same amount of computation time.

2.4.4 Branching strategies

At each branch-and-bound node, we achieve an optimal solution of the LMP using the column generation procedure and the separation algorithm; the value of this solution is a lower bound at that node. If this lower bound is not less than the current upper bound, the associated node is pruned; otherwise, branching must take place. If the optimal solution of the LMP is integral and the optimal solution value is less than the current upper bound, we update the upper bound. The integral optimal solution of the LMP must contain only acyclic routes even if routes with cycles are allowed. This is because we can always find acyclic routes to replace the routes with cycles in an integral feasible solution and generate another integral feasible solution with less objective value. However, it should be noted that the fractional optimal solution of the LMP may include routes with cycles if they are allowed.

As explained in Desaulniers et al. [40], we can hardly branch on master problem variables $\theta_r$ since fixing such variables at 0 requires preventing label-setting algorithms from generating the corresponding routes, which significantly increases the complexity of solving the pricing subproblem. Therefore, it is better to choose branching strategies compatible with the algorithms for the pricing subproblems, i.e., the pricing subproblems at the nodes resulting from such branchings could be solved in a way similar to the one used at their parent node. This requires
that branching constraints do not change the structure of the pricing subproblem. In our branch-and-price-and-cut algorithm, we choose two branching strategies, namely branching on the number of vehicles and on arcs.

**Branching on the number of vehicles.** The number of vehicles used in the optimal solution of the LMP can be calculated by $m' = \sum_{r \in \Omega} \theta_r$. If this value is fractional, we branch the current node into two child nodes with constraints $\sum_{r \in \Omega} \theta_r \leq \lfloor m' \rfloor$ and $\sum_{r \in \Omega} \theta_r \geq \lceil m' \rceil$, respectively. The pricing subproblems at these two child nodes have the same structure as that at their parent node.

**Branching on arcs.** We next branch on the edge $(i, j)$ with fractional flow $\hat{x}_{i,j}$ that is farthest to an integer. Two child nodes are created by fixing $x_{i,j} = 1$, implying that edge $(i, j)$ must be used, and $x_{i,j} = 0$, implying that edge $(i, j)$ is forbidden. In the former case, we delete from $E$ all edges $(i, j')$ and $(i', j)$ with $j' \neq j$ and $i' \neq i$, and remove from the RLMP all variables $\theta_r$ and columns associated with routes that contain at least one of the above deleted edges. In the latter case, we delete edge $(i, j)$ from $E$ and remove from the RLMP all variables $\theta_r$ and columns associated with routes that contain edge $(i, j)$. Obviously, this type of branching only alters the structure of the underlying graph and removes some of variables from the RLMP, which does not require to modify the label-setting algorithm when solving the pricing subproblems at the child nodes.

### 2.5 Computational experiments

We implemented two column generation procedures; the first one is called CG1 that employs the tabu search column generator and the BBLS algorithm with decremental state-space relaxation, and the second one is called CG2 that only employs the BBLS algorithm with state space relaxation. Specifically, CG1 invokes the BBLS algorithm only when the tabu search algorithm fails to identify negative reduced cost columns. In this section, we only report and analyze
the results of two branch-and-price-and-cut implementations that employs CG1 and CG2, which are called BPC1 and BPC2, respectively. Both of the BPC1 and BPC2 outperform the branch-and-price-and-cut implement that utilizes the BBLS algorithm without using any accelerating strategy.

2.5.1 Instance generation

We conducted experiments on a data set derived from six TSP instances *fnl4461, brd14051, d15112, d18512, nrw1379 and pr1002* of TSPLIB [105]. For each of these TSP instances, ten subsets of \( n + 1 \) vertices were randomly selected, where \( n = 29, 39 \) and 49. One arbitrary vertex was designated as the depot. Each instance group is identified by the name of its TSP instance and the number of vertices; for example, instance group *brd14051-30* contains ten 30-vertex instances generated from TSP instance *brd14051*. We denote by \( d_{\text{max}} = \max_{i \in V_C} \{d_{0,i} + d_{i,n+1}\} \) the greatest distance for any route involving a single customer and imposed on each vehicle a travel distance limit \( L = 2.0 \times d_{\text{max}} \). Next, we performed the CI heuristic to construct an initial solution for each vertex set and recorded the number of used vehicles as \( K_{\text{ini}} \). Finally, we set the number of available vehicles to \( K = \max \left\{ K_{\text{ini}}, \left\lceil n/5 \right\rceil \right\} \), thereby generating 180 test instances. All instances as well as detailed solutions are available in the online supplement at: [www.computational-logistics.org/orlib/mtrpd](http://www.computational-logistics.org/orlib/mtrpd)

2.5.2 Experimental setup

All algorithms were coded in Java and all experiments were conducted on a Dell server with an Intel Xeon E5520 2.26 GHz CPU, 8 GB RAM and Linux operating system. The linear programming model RLMP was solved by simplex algorithm implemented by ILOG CPLEX 12.0. Computation times reported are in CPU seconds on this server.
We imposed a time limit of 10,800 seconds on each run of the branch-and-price-and-cut algorithm. The parameters used in this chapter were fixed as: \( \xi = 10 \), \( maxIter = 200 \) and \( maxCol = 100 \). According to some preliminary experiments, we observed that only the value of \( \xi \) affected the performance of the proposed tabu search algorithm while all these parameters had little impact on the performance of the branch-and-price-and-cut algorithm.

### 2.5.3 Results and analysis

We solved all instances using both BPC1 and BPC2, and report the computational results in Tables 2.1 – 2.3. At the root node, we first solved the LMP without the introduction of any violated SE cut using both CG1 and CG2; the optimal linear solution values are presented in columns “LP” under both the blocks “BPC1” and “BPC2”. Subsequently, we added the violated SE cuts into the LMP, and performed the column generation procedure and the separation algorithm alternatively. We find from the computational results that the violated SE cuts can be identified only when using CG2. Therefore, under the block “BPC2” we present the improved lower bound and the number of violated SE cuts at the root node in column “LPC” and “Cuts”. The values of the optimal integer solutions achieved by the BPC1 and BPC2 are reported in columns “IP”. The values in columns “LP Gap (%)” and “LPC Gap (%)” were calculated by \( (IP - LP) / IP \) or \( (IP - LPC)/IP \). The columns “Nodes” give the number of nodes explored by the branch-and-price-and-cut algorithm. The computation times for “LP” and “IP” are provided in columns “LP Time” and “IP Time”, respectively.

As shown in Table 3.3, BPC1 failed to achieve the optimal solutions for instances brd14051-50-3 and nrw1379-50-1, while BPC2 only failed to obtain the optimal solution for instance nrw1379-50-1. In the last row of these tables, we report the average values of “LP Time”, “LP Gap (%)”, “Nodes” and “IP Time” and “LPC
### Table 2.1: Performance comparison between BPC1 and BPC2 on 30-vertex instances.

As shown in columns “LP” or “LP Gap (%),” the lower bounds achieved by CG1 is better than or equal to those obtained by CG2; the difference of their average gaps over all solved instances is 0.96%. However, columns “LP Time” show that CG2 consumed less or equal computation times for all instances, compared to CG1. The use of the SE cuts improved the lower bounds generated by CG2 since the average gap was reduced from 1.16% to 0.53%. The lower bounds produced by CG2 with the help of SE cuts are still not better than those produced by CG1; the difference of their average gaps is 0.32%. As mentioned in...
Table 2.2: Performance comparison between BPC1 and BPC2 on 40-vertex instances.
<table>
<thead>
<tr>
<th>Instance</th>
<th>BPC1 LP Time</th>
<th>BPC2 LP Time</th>
<th>BPC1 Gap(%)</th>
<th>BPC2 Gap(%)</th>
<th>Nodes Cuts IP Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>brd14051-50</td>
<td>0.00</td>
<td>133,587.1</td>
<td>58</td>
<td>0.04</td>
<td>9</td>
</tr>
<tr>
<td>d15112-50</td>
<td>0.00</td>
<td>351,861.7</td>
<td>44</td>
<td>0.51</td>
<td>175</td>
</tr>
<tr>
<td>d18512-50</td>
<td>0.00</td>
<td>131,512.3</td>
<td>87</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>fnl4461-50</td>
<td>0.00</td>
<td>81,539.0</td>
<td>65</td>
<td>0.03</td>
<td>3</td>
</tr>
<tr>
<td>nrw1379-50</td>
<td>0.00</td>
<td>39,146.0</td>
<td>27</td>
<td>0.15</td>
<td>13</td>
</tr>
<tr>
<td>pr1002-50</td>
<td>0.00</td>
<td>272,547.5</td>
<td>22</td>
<td>0.13</td>
<td>9</td>
</tr>
<tr>
<td>Average</td>
<td>37.1</td>
<td>0.19</td>
<td>25.9</td>
<td>337.0</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Table 2.3: Performance comparison between BPC1 and BPC2 on 50-vertex instances.
Section 2.3.3.3, the weaker lower bounds usually necessitate generation of more branch-and-bound nodes, which is clearly indicated by the comparison of columns “Nodes”.

We obtained the optimal solutions for all instances except instance nrw1379-50-1. The problem difficulty increases with the number of vertices, which is revealed by the average computation times. On average, BPC2 used less computation times than BPC1. We sorted the “LP Gap (%)” values in the block “BPC1” and “LPC Gap (%)” values in the block “BPC2” in ascending order, and then plotted the results in Figures 2.2 and 2.3. These figures reveal that BPC1 and BPC2 produced high-quality lower bounds at the root node. With the help of such high-quality lower bounds, most test instances were solved to optimality after exploring a small number of branch-and-bound nodes. The statistical results on the numbers of branch-and-bound nodes generated by BPC1 and BPC2 are shown in Figure 2.4. From this figure, we can see that 28% and 6% of instances were optimally solved by BPC1 and BPC2 with only one branch-and-bound node.

![Figure 2.2](image-url)

Figure 2.2: The sorted “LP Gap (%)” values associated with the BPC1.
Figure 2.3: The sorted “LPC Gap (%)” values associated with the BPC2.

Figure 2.4: The statistical results on the numbers of branch-and-bound nodes generated by BPC1 and BPC2.

2.6 Conclusions

In this chapter, we propose an exact branch-and-price-and-cut algorithm for solving the MTRPD, which is a natural extension to the MTRP that considers a limit to the travel distance of the vehicle. The key component in the branch-and-price-and-cut algorithm is the algorithm used to solve the pricing subproblem, which is an elementary resource-constrained shortest path problem with cumulative cost. We have obtained the optimal solutions for all but one instance within 10,800
seconds. The branch-and-price-and-cut implementation with state-space relaxation consumed less computation time than the other implementation with tabu search column generator and decremental state-space relaxation. The experimental results and analysis presented in this study serve as benchmarks for future researchers. Our work is the first attempt at designing bounded bidirectional label-setting algorithm for the pricing problems with cumulative cost. Future research may consider either studying other exact algorithms to improve our results for the MTRPD or designing exact algorithms for other routing problems with cumulative cost.
Chapter 3

Branch-and-price-and-cut for the Split-collection Vehicle Routing Problem with Time Windows and Linear Weight-related Cost

3.1 Introduction

The vehicle routing problem (VRP) and its variants have been extensively studied in literature [125]. These problems consist of designing a set of least-cost routes that fulfill all customer demands and respect a group of operational constraints, such as vehicle capacity, route duration and time windows. The vast majority of the existing vehicle routing models assume that the cost of traversing a route equals the length of that route; a common objective for such models is to minimize the total traveling distance. However, in most practical logistical problems, the real transportation cost depends on many other factors apart from traveling distance, such as vehicle weight, vehicle speed, road conditions and fuel price. Consequently, the distance-minimization vehicle routing models cannot be directly applied by the industrial practitioners that wish to minimize their total
transportation costs.

In this chapter, we only take the effect of vehicle weight and traveling distance on transportation cost into consideration and assume other factors are unchanged. As a result, the transportation cost can be calculated by $d \times f(w)$, where $d$ is the traveling distance and $f(w)$ is a function representing the cost per unit distance paid by the vehicle with weight $w$. Ignoring the effect of vehicle weight is equivalent to setting $f(w)$ to be a constant for any $w > 0$. This weight-related cost might have a great impact on the sequencing of the customers along the routes, which is illustrated by the example shown in Figure 3.1. This example involves four vertices, i.e., customers $A, B, C$ and depot $D$, whose locations are vertices of a square with side length 1. It is assumed that $f(w) = 0.08w$, the curb weight of a vehicle is 5, and the weight demands at customers $A, B$ and $C$ are 1, 15 and 1, respectively. Figure 3.1(a) indicates a shortest route that incurs a cost of 4.32 while Figure 3.1(b) shows a least-cost route with a cost of 4.12. By this example, we can observe that when the weight-related cost is imposed, the customers with more weight demands tend to be served with higher priority.

![Figure 3.1: The impact of vehicle weight on customer sequence.](image)

Numerous real-life applications of the vehicle routing models with weight-related costs arise naturally in Chinese expressway transportation system. As of the end of 2012, over twenty five Chinese provinces have implemented toll-by-weight schemes in which expressway toll per unit distance is levied according to
a monotonically increasing function $f(w)$. Under such toll schemes, a vehicle is charged quite differently when it is empty, normally loaded and overloaded. Moreover, we can also find applications from the transportation service providers who are concerned with fuel consumption and the environmental impact of greenhouse gas (GHG) emissions. The fuel expenditure accounts for a large portion of the overall transportation cost and thus greatly affects the profits of transportation service providers [127]. The fuel consumption rate is directly related to vehicle weight; for example, for a vehicle of some type, its fully loaded status might consume more than twice as much diesel fuel as its empty status. In the last decade, the hazardous impacts of GHG, which is directly related to the consumption of fossil fuel, have received growing concerns from the public. Transport sector is one of the key sources of GHG emissions. As revealed by U.S. Greenhouse Gas Inventory Report published in 2012, transportation activities account for 32% of U.S. CO$_2$ emissions from fossil fuel combustion in 2010. There is a clear tendency that transportation service providers will be forced to undertake the cost of their GHG emissions in the context of new regulations. The cost of fuel consumed or GHG emitted per unit distance by a vehicle with weight $w$ can be represented by a function $f(w)$.

In existing literature, we can only find several prior studies on the vehicle routing models that take vehicle weight into account. The VRP with toll-by-weight scheme was first mentioned by Shen et al. [119], who integrated toll-by-weight schemes into the traditional capacitated VRP and developed a simulated annealing algorithm to solve the problem. More recently, Zhang et al. [131] tackled a vehicle routing model that involves toll-by-weight schemes and a single vehicle using a branch-and-bound algorithm. However, this algorithm cannot be adapted to solve the problem with multiple vehicles. Zhang et al. [130] proposed a multi-depot VRP in which a constant surcharge $C_l$ is incurred for per unit
distance per unit weight. Although they did not mention the applications of their problem in the context of Chinese expressway transposition system, imposing this weight-related surcharge is essentially equivalent to levying tolls according to a linear function \( f_1(w) = C_l \times w \). The objective of their problem is to minimize the total transportation cost, consisting of distance-related cost, weight-related cost and the fixed cost of dispatching each vehicle. They implemented a scatter search algorithm to solve their problem.

The influence of vehicle weight is perceived in several vehicle routing models that incorporate the costs of fuel and GHG emissions in their objective functions. Most of these models focus on analyzing the influence of vehicle speed, and/or vehicle weight; therefore we can roughly divide them into three classes. The first class of articles only discussed the relationship between vehicle speed and GHG emissions (i.e., ignore the influence of vehicle weight), and investigated methodologies to determine both the route and speed of each vehicle for minimal fuel and emission costs; representative examples include Palmer [97], Figliozzi [50], Jabali et al. [71].

The articles in the second class only took vehicle weight into consideration by assuming vehicle speed to be constant. The seminal work of the vehicle routing models that relate vehicle weight to fuel consumption was conducted by Kara et al. [73], who introduced an Energy-Minimizing VRP. In this problem, the energy consumed for traversing an edge equals the product of the vehicle weight and the edge length, and the objective is to minimize the total energy rather than the total traveling distance. Lately, Huang et al. [66] proposed a variant of the VRP with Simultaneous Pickups and Deliveries (VRPSPD) that incorporates the cost of fuel consumption and carbon emissions in the objective function. They assumed without proof that the fuel consumption and carbon emissions per unit distance are both linearly directly proportional to the vehicle weight.
Both Kara et al. [73] and Huang et al. [66] formulated their problems into mixed integer programming (MIP) models and then solved the models using off-the-shelf MIP solvers. Based on some statistical data, Xiao et al. [127] derived that the fuel consumption rate can be approximated to a linear function of vehicle weight. They proposed a string-model-based simulated annealing algorithm with a hybrid exchange rule to solve both the distance-minimization VRP and the fuel-minimization VRP. Their experiments on 27 benchmark instances show that the fuel-minimization VRP could help reduce fuel consumption by 5% on average, compared to the corresponding distance-minimization VRP.

The articles in the last class tackled more general and practical vehicle routing models, where the cost of fuel consumption and GHG emissions is a function of vehicle speed and vehicle weight. The first such model was introduced by Kuo [78], who built a fuel-minimization vehicle routing model on the time-dependent VRP (TDVRP) [68, 79]. In the TDVRP, the time horizon is discretized into a number of intervals. For each edge and each time interval, there is a fixed and known travel speed for all vehicles. The objective of the TDVRP is to minimize the total travel times of all vehicles. In Kuo [78], the authors modified the TDVRP by replacing minimizing total travel time with minimizing total fuel consumption. The miles per gallon (MPG) and the gallons per hour (GPH) for an empty vehicle traversing each edge in each time interval are input parameters. They assumed that the fuel consumption rate increases linearly with vehicle weight. For a given routing plan, the total fuel consumed can be easily calculated with the information of MPG, GPH and the vehicle weight on each edge. A simulated annealing algorithm was developed to solve the fuel-minimization TDVRP. Bektaş and Laporte [16] presented a Pollution-Routing Problem (PRP), which is an extension of the classical VRP with more comprehensive objective function that accounts for the costs of driving, GHG emissions and fuel. The driving cost is
linearly directly proportional to the total travel time of all vehicles. The amount of fuel consumed on an edge is approximated as \((\alpha w + \beta v^2) \times d\), where \(\alpha\) is an edge-specific constant, \(w\) is the vehicle weight, \(\beta\) is a vehicle-specific constant, \(v\) is the vehicle speed and \(d\) is the edge length. It can be easily observed that when the vehicle speed is fixed, the amount of fuel consumed per unit distance is a linear function of the vehicle weight. They formulated the PRP into an integer linear programming model, where the vehicle speed associated with each edge is a decision variable, and then applied CPLEX 12.1 with default settings to solve the model.

The aim of this chapter is to address a problem extended from the split-delivery VRP with time windows (SDVRPTW) by modeling the cost per unit distance as a linear function of the vehicle’s load weight \(w\), i.e., \(f(w) = a \times w + b\), where \(a\) and \(b\) are constant. The SDVRPTW is adapted from the well-studied vehicle routing problem with time windows (VRPTW) by allowing customer demands to exceed the vehicle capacity and relaxing the constraint that each customer must be visited exactly once. We refer the reader to Ho and Haugland [65], Desaulniers [39], Archetti et al. [8] for more details of the SDVRPTW. The SDVRPTW can be used to model the cases of delivering goods to or collecting goods from customers. In this article, we consider the collection case and therefore our problem is called the *split-collection vehicle routing problem with time windows and linear weight-related cost (SCVRPTWL)*. The SDVRPTW is a special case of the SCVRPTWL with \(a = 0\) and \(b = 1\). Since the combination of several linear functions is also linear, the linear weight-related cost function can be used to model the applications with one or several cost factors, such as traveling distance, linear tolls, fuel consumption and GHG emissions.

The main contributions are summarized as follows. First, we introduce a more practical and general vehicle routing model that considers the vehicle weight.
Second, we provide a branch-and-price-and-cut algorithm for the problem with any type of linear weight-related cost function. In this algorithm, the linear relaxation of the master problem at each branch-and-bound node is solved using a column generation procedure [41]. Although the master problem of the SCVRPTWL is similar to that of the SDVRPTW presented in Desaulniers [39], our pricing subproblem is more complicated compared to that of the SDVRPTW. Thus, we designed a tailored label-setting algorithms to solve the pricing problem of the SCVRPTWL. The dominance procedures used in most of the existing label-setting algorithms are usually based on comparing two labels. However, our label-setting algorithm employs a novel and more efficient dominance procedure that checks whether a label is dominated by a set of labels. Several techniques such as the tabu column generator, decremental search, and bi-directional search are proposed to accelerate the column generation procedure. Third, our comprehensive experimental results show the effectiveness of our proposed algorithm and serve as a baseline for future researchers working on this and other related problems.

3.2 Problem Description, Properties and Formulation

The SCVRPTWL is defined on a directed graph $G = (V, E)$, where $V = \{0, 1, \ldots, n, n+1\}$ is the vertex set and $E = \{(i, j) | i, j \in V, i \neq j, i \neq n+1, j \neq 0\}$ is the edge set. Vertices 0 and $n+1$ are known as the exit from and the entrance to the depot, respectively, and the set of remaining vertices $V_C = \{1, \ldots, n\}$ denotes the set of $n$ customers. Each vertex $i$ is characterized by a positive weight demand $d_i$, a service time $s_i$, and a time window $[e_i, l_i]$ within which the service can be started. For notational convenience, we set $d_0 = 0$, $d_{n+1} = +\infty$, $s_0 = s_{n+1} = 0$, $e_0 = 0$, and $l_0 = \infty$. The service times $s_i$ are assumed to be non-negative for all $i \in V_C$, and the time windows are non-overlapping for all $i \in V_C$. Each edge $(i, j)$ has a distance $c_{ij}$ and a travel time $t_{ij}$, and the service times are non-negative. The demand of each customer is non-negative, and the problem is constrained to ensure that the total demand of each route does not exceed the capacity of the depot.

The objective is to minimize the total travel cost, subject to the constraints that each customer is visited exactly once, the time windows are respected, and the capacities of the depot are not exceeded. The problem is known to be NP-hard, and several heuristic and exact algorithms have been proposed to solve it. The SL04B algorithm, for example, is a branch-and-price-and-cut algorithm that is designed to solve the problem efficiently. The algorithm uses a column generation procedure to solve the master problem at each branch-and-bound node, and the pricing subproblem is solved using a tailored label-setting algorithm. The dominance procedures used in most of the existing label-setting algorithms are usually based on comparing two labels. However, our label-setting algorithm employs a novel and more efficient dominance procedure that checks whether a label is dominated by a set of labels. Several techniques such as the tabu column generator, decremental search, and bi-directional search are proposed to accelerate the column generation procedure. Third, our comprehensive experimental results show the effectiveness of our proposed algorithm and serve as a baseline for future researchers working on this and other related problems.
\( e_0 = e_{n+1} = 0 \) and \( l_0 = l_{n+1} = +\infty \). Each edge \((i, j) \in E\) has a nonnegative distance \( c_{i,j} \) and a nonnegative traversing time \( t_{i,j} \). We assume that both the distances and traversing times satisfy the triangle inequality. We denote by 
\[ V^+(i) = \{ j \in V | e_i + s_i + t_{i,j} \leq l_j, (i, j) \in E \} \]
and 
\[ V^-(i) = \{ j \in V | e_j + s_j + t_{j,i} \leq l_i, (j, i) \in E \} \]
the vertices immediately succeeding and preceding vertex \( i \) on graph \( G \).

We are given an unlimited number of homogeneous vehicles each with a weight capacity \( Q \). Each vehicle is allowed to perform a collection pattern, i.e., it starts from vertex 0, visits a subset of customers, collects some quantity of products at each visited customer and returns to vertex \( n+1 \). A collection pattern is defined as a route with specified collected quantity at each vertex. If a vehicle arrives at vertex \( i \) prior to \( e_i \), it must wait until \( e_i \) and then starts the service. A collection pattern is feasible if its associated route respects the time windows of all visited customers and its total collected demand does not exceed \( Q \). The demand of each customer may be fulfilled by multiple vehicles, i.e., the customer demand may be greater than the vehicle capacity and a customer is allowed to be visited more than once. The traversal cost of edge \((i, j)\) paid by the vehicle with load weight \( w_{i,j} \) is calculated by \( c_{i,j} \times f(w_{i,j}) \), where \( f(w_{i,j}) = a \times w_{i,j} + b \) and the intercept \( b \) is the cost incurred by the curb weight of the vehicle. The objective of the SCVRPTWL is to find a set of feasible collection patterns such that all customer demands are fulfilled and the total traversal cost is minimized.

We can easily observe a property (called Property 1) that there must exist an optimal solution in which each route visits each customer at most once. In Desaulniers [39], the authors presented a theorem regarding the optimal solutions to the SDVRPTW. We first show that this theorem is also valid for the SCVRPTWL and then derive another two properties. All these three properties can help reduce the search space of the SCVRPTWL significantly.
Theorem 3.1  Given an instance of the SCVRPTWL where the matrices \([c_{i,j}]\) and \([t_{i,j}]\) satisfy the triangle inequality, there must exist an optimal solution to this instance in which no two vehicles have more than one split customer in common.

Proof:  Proof. Suppose there exist two collection patterns \(p_1\) and \(p_2\) that have two common customers \(i\) and \(j\). The quantities collected at customers \(i\) and \(j\) in pattern \(p_1\) (respectively, \(p_2\)) are \(\delta_1^i\) and \(\delta_2^i\) (respectively, \(\delta_1^j\) and \(\delta_2^j\)). Note that \(\delta_1^i + \delta_2^i \leq d_i\) and \(\delta_1^j + \delta_2^j \leq d_j\). We increase \(\delta_1^i\) and \(\delta_2^j\) by \(\epsilon\), decrease \(\delta_1^j\) and \(\delta_2^i\) by \(\epsilon\) (see Figure 3.2), and do not change the quantities collected at the remaining customers. Obviously, only the costs of edges between customers \(i\) and \(j\) may be affected by this quantity adjustment. Let \(c_{i \rightarrow j}^1\) and \(c_{i \rightarrow j}^2\) be the traveling distances from customer \(i\) to customer \(j\) in patterns \(p_1\) and \(p_2\), respectively (note that \(i \rightarrow j\) may cover more than two vertices). After the adjustment, the cost of \(p_1\) will increase by \(a \times c_{i \rightarrow j}^1 \times \epsilon\) while the cost of \(p_2\) will decrease by \(a \times c_{i \rightarrow j}^2 \times \epsilon\). If \(a \times \epsilon \times (c_{i \rightarrow j}^1 - c_{i \rightarrow j}^2) \leq 0\), we have a motivation to increase \(\delta_1^i\) until either \(\delta_j^1 = 0\) or \(\delta_i^2 = 0\) for less total cost. If \(\delta_j^1 = 0\) or \(\delta_i^2 = 0\), we can safely remove customer \(j\) from \(p_1\) or customer \(i\) from \(p_2\) without increasing the costs of pattern \(p_1\) or \(p_2\). We can analyze the case when \(c_{i \rightarrow j}^1 \geq c_{i \rightarrow j}^2\) in the same manner. Hence, if \(p_1\) and \(p_2\) exist in an optimal solution, they can be adjusted to have one customer in common without increasing the total cost. □

The properties derived from Theorem 3.1 are: there must exist an optimal solution to the SCVRPTWL in which

- at most one vehicle assigned to a route with two or more customers (Property 2).
- each edge \((i, j) \in E_C\) appears at most once, where \(E_C = \{(i, j) | (i, j) \in E, i, j \in V_C\}\) (Property 3).
We now present an arc-flow formulation for the SCVRPTWL, which will be exploited in the Dantzig-Wolfe decomposition proposed in the next section. This formulation uses the following additional notations:

**Parameters**

- $K = \{1, 2, \ldots, m\}$: the set of $m$ available vehicles.
- $M$: a sufficiently large positive number.

**Decision Variables**

- $x_{i,j,k}$: the binary variable that equals 1 if vehicle $k$ traverses edge $(i, j)$, and 0 otherwise.
- $w_{i,j,k}$: the load weight of vehicle $k$ who traverses edge $(i, j)$.
- $z_{i,j,k}$: the cost paid by vehicle $k$ for traversing edge $(i, j)$.
- $a_{i,k}$: the service starting time of vehicle $k$ at customer $i$. 

Figure 3.2: An example of two patterns that have two customers in common.


Using these notations, the SCVRPTWL can be modeled as:

\[
\min \sum_{k \in K} \sum_{i \in V_C \cup \{0\}} \sum_{j \in V^+(i)} z_{i,j,k} \tag{3.1}
\]

s.t. \( z_{i,j,k} \geq c_{i,j}(aw_{i,j,k} + bx_{i,j,k}), \forall k \in K, \ i \in V_C \cup \{0\}, \ j \in V^+(i) \tag{3.2} \)

\[
\sum_{k \in K} \left( \sum_{j \in V^+(i)} w_{i,j,k} - \sum_{j \in V^-(i)} w_{j,i,k} \right) \geq d_i, \ \forall \ i \in V_C \tag{3.3}
\]

\[
\sum_{k \in K} \sum_{j \in V^+(i)} x_{i,j,k} \geq \left\lceil \frac{d_i}{Q} \right\rceil, \ \forall \ i \in V_C \tag{3.4}
\]

\[
\sum_{i \in V_C \cup \{n+1\}} x_{0,i,k} = 1, \ \forall \ k \in K \tag{3.5}
\]

\[
\sum_{j \in V^+(i)} x_{i,j,k} = \sum_{j \in V^-(i)} x_{j,i,k} \leq 1, \ \forall \ k \in K, \ i \in V_C \tag{3.6}
\]

\[
\sum_{i \in V_C \cup \{0\}} x_{i,n+1,k} = 1, \ \forall \ k \in K \tag{3.7}
\]

\[
w_{i,j,k} \leq Qx_{i,j,k}, \ \forall \ k \in K, \ i \in V_C \cup \{0\}, \ j \in V^+(i) \tag{3.8}
\]

\[
a_{j,k} \geq a_{i,k} + s_i + t_{i,j} + M(x_{i,j,k} - 1), \ \forall \ k \in K, \ i \in V_C \cup \{0\}, \ j \in V^+(i) \tag{3.9}
\]

\[
e_i \leq a_{i,k} \leq l_i, \ \forall \ k \in K, \ i \in V \tag{3.10}
\]

\[
x_{i,j,k} \in \{0,1\}, \ \forall \ k \in K, \ i \in V_C \cup \{0\}, \ j \in V^+(i)
\]

\[
a_{i,k} \geq 0, \ \forall \ k \in K, \ i \in V
\]

\[
z_{i,j,k} \geq 0, \ w_{i,j,k} \geq 0, \ \forall \ k \in K, \ i \in V_C \cup \{0\}, \ j \in V^+(i)
\]

The objective function (3.1) aims at minimizing the total travel distance. The traversal cost of edge \((i, j)\) incurred by the vehicle with load weight \(w_{i,j}\) is calculated by Constraints (3.2). Constraints (3.3) ensure that the demand of each customer is fulfilled. A minimum number of vehicles that serve customer \(i\) is imposed by Constraints (3.4), which are redundant constrains that are used to strengthen the linear relaxation of the model. Constraints (3.5) – (3.7) define the
structure of each possible route. The load weight of each vehicle on each edge cannot exceed $Q$ and thus Constraints (3.8) apply. Constraints (3.9) and (3.10) ensure that all customer time windows are respected.

### 3.3 Dantzig-Wolfe Decomposition

We can directly apply CPLEX to handle the arc-flow formulation. Nevertheless, after some preliminary experiments, we find that the size of the instances optimally solved by CPLEX is quite limited. To achieve optimal solutions for instances of practical size, we reformulate the SCVRPTWL into a *master problem* through Dantzig-Wolfe decomposition [36] and then develop a branch-and-price-and-cut algorithm to solve it.

Applying Dantzig-Wolfe decomposition to the arc-flow formulation yields a master problem and a *pricing subproblem*. The master problem contains a decision variable for each collection pattern. We solve the master problem by a branch-and-bound procedure, where at each branch-and-bound node a lower bound is obtained by column generation procedure and the introduction of violated valid inequalities. The pricing subproblem is solved using a tailored label-setting algorithm.

#### 3.3.1 Master Problem

To present the master problem, we define the following additional notations:

**Parameters**

- $R^s$: the set of all routes visiting a single customer and satisfying the time window constraint.

- $R^m$: the set of all routes visiting more than one customer and satisfying all time window constraints.
• \( P_r \): the set of all collection patterns compatible with route \( r \).

• \( c_{r,p} \): the cost of collection pattern \( p \), where \( p \in P_r \).

• \( \alpha_{i,r} \): the binary parameter that equals 1 if customer \( i \) is used in route \( r \), and 0 otherwise.

• \( \beta_{i,j,r} \): the binary parameter that equals 1 if edge \((i,j)\) is used in route \( r \), and 0 otherwise.

• \( \delta_{i,p} \): the quantity collected at customer \( i \) in pattern \( p \).

**Decision variables**

• \( \theta_{r,p} \), the nonnegative integer variable indicating the number of the vehicles assigned to pattern \( p \) compatible with route \( r \).

• \( \theta_r \), the nonnegative integer (respectively, binary) variable indicating the number of the vehicles assigned to route \( r \in R^s \) (respectively, \( R^m \)). The binary requirement is derived from Property 2.

With the above notations, the master problem (MP) is given as:

\[
\begin{align*}
z^{MP} &= \min \sum_{r \in R} \sum_{p \in P_r} c_{r,p} \theta_{r,p} \\
\text{s.t.} & \quad \sum_{r \in R} \sum_{p \in P_r} \delta_{i,p} \theta_{r,p} \geq d_i, \quad \forall \ i \in V_C \\
& \quad \sum_{r \in R} \sum_{p \in P_r} \alpha_{i,r} \theta_{r,p} \geq \left\lceil \frac{d_i}{Q} \right\rceil, \quad \forall \ i \in V_C \\
& \quad \theta_{r,p} \geq 0, \ \forall \ r \in R, \ p \in P_r \\
& \quad \theta_r = \sum_{p \in P_r} \theta_{r,p}, \ \forall \ r \in R \\
& \quad \theta_r \in \{0,1\}, \ \forall \ r \in R^m \\
& \quad \theta_r \ \text{integer,} \ \forall \ r \in R^s
\end{align*}
\]
The objective function (3.11) aims at minimizing the total travel distance. Constraints (3.12) – (3.13) are equivalent to Constraints (3.3) – (3.4), respectively. Constraints (3.13) are redundant constraints that are used to strengthen the linear relaxation of the MP (called LMP for short). Constraints (3.14) – (3.17) are binary or integrality requirements on the decision variables $\theta_{r,p}$ and $\theta_r$. In this formulation, each variable $\theta_{r,p}$ corresponds to a column composed of parameters $c_{r,p}, \delta_{i,p}$ and $\alpha_{i,r}$.

In practice, even for a small-size instance, the master problem contains a huge number of variables (or columns). Hence, this model cannot be directly handled by CPLEX. With a subset of variables $\theta_{r,p}$, the optimal solution of the LMP can be obtained with the help of the column generation procedure. Therefore, we do not need to enumerate all variables $\theta_{r,p}$ explicitly. Variables $\theta_r$ are not required in the LMP but will be used to check whether the optimal solution to the LMP is also optimal to the MP.

### 3.3.2 Pricing Subproblem

Given a dual solution to the LMP, the pricing subproblem is used to find a master variable $\theta_{r,p}$ (i.e., a collection pattern $p$ compatible with route $r$) that has the least reduced cost. Solving the pricing subproblem is essentially equivalent to enumerating all feasible collection patterns. Below we will use $\pi = (\pi_1, \ldots, \pi_n)$ and $\mu = (\mu_1, \ldots, \mu_n)$ as the values of the dual variables associated with Constraints (3.12) – (3.13), respectively, and set $\pi_0 = \pi_{n+1} = 0$ and $\mu_0 = \mu_{n+1} = 0$ without loss of generality. The reduced cost of a pattern $p$ compatible with route $r$ can be calculated by:

$$
\bar{c}_{r,p} = c_{r,p} - \sum_{i \in r} (\delta_{i,p} \pi_i + \mu_i)
$$

(3.18)
Since all vehicles are identical, the pricing subproblem associated with each vehicle can be written as:

\[
 z^{PS} = \min \sum_{(i,j) \in E} c_{i,j}(aw_{i,j} + bx_{i,j}) - \sum_{i \in V_C} \pi_i \delta_i - \sum_{i \in V_C} \mu_i \sum_{j \in V^+(i)} x_{i,j} \tag{3.19}
\]

s.t.

\[
 \sum_{j \in V^+(0)} x_{0,j} = 1 \tag{3.20}
\]

\[
 \sum_{i \in V^{-(n+1)}} x_{i,n+1} = 1 \tag{3.21}
\]

\[
 \sum_{j \in V^+(i)} x_{i,j} = \sum_{j \in V^-(i)} x_{j,i}, \ \forall \ i \in V_C \tag{3.22}
\]

\[
 \delta_i = \sum_{j \in V^+(i)} w_{i,j} - \sum_{j \in V^-(i)} w_{j,i}, \ \forall \ i \in V_C \tag{3.23}
\]

\[
 \delta_i \leq \min\{d_i, Q\} \sum_{j \in V^+(i)} x_{i,j}, \ \forall \ i \in V_C \tag{3.24}
\]

\[
 w_{i,j} \leq Q x_{i,j}, \ \forall \ i \in V_C \cup \{0\}, \ j \in V^+(i) \tag{3.25}
\]

\[
 a_j \geq a_i + s_i + t_{i,j} + M(x_{i,j} - 1), \ \forall \ i \in V_C \cup \{0\}, \ j \in V^+(i) \tag{3.26}
\]

\[
 e_i \leq a_i \leq l_i, \ \forall \ i \in V_C \tag{3.27}
\]

\[
 x_{i,j} \in \{0, 1\}, \ w_{i,j} \geq 0, \ \forall \ i \in V_C \cup \{0\}, \ j \in V^+(i) \tag{3.28}
\]

\[
 a_i \geq 0, \ \delta_i \geq 0, \ \forall \ i \in V_C \tag{3.29}
\]

where

- \( x_{i,j} \): the binary variable that equals 1 if edge \((i, j)\) is used in the pattern, and 0 otherwise.

- \( w_{i,j} \): the load weight on edge \((i, j)\).

- \( a_i \): the service starting time at customer \(i\).

- \( \delta_i \): the quantity collected at customer \(i\).

The objective function (3.19) aims to achieve the minimal reduced cost of all
feasible collection patterns. Constraints (3.20) and (3.21) require that the route must start from vertex 0 and end at vertex \( n + 1 \). Constraints (3.22) ensure that each customer can be visited at most once. Constraints (3.23) and (3.24) state that the quantity collected at customer \( i \) cannot exceed \( d_i \). Constraints (3.25) guarantee that the flow on each edge cannot exceed the vehicle capacity and equals zero if that edge is not used. Constraints (3.26) define the relationship between the service starting times of two consecutively visited customers. Constraints (3.27) ensure that the time windows of all visited customers must be respected.

In many existing branch-and-price algorithms that were developed to solve the VRPTW or other vehicle routing problems, their pricing subproblems are usually elementary shortest path problem with resource constraints (ESPPRC) [49]. Examples can be found in Desrochers et al. [43], Gutiérrez-Jarpa et al. [62], Azi et al. [11] and Bettinelli et al. [22]. Evidently, it is not appropriate to view our pricing subproblem as an ESPPRC due to the existence of variables \( w_{i,j} \). We call our pricing subproblem elementary least-cost path problem with resource constraints (ELPPRC). Similar pricing subproblems can be found in Ioachim et al. [69] and Ribeiro et al. [107]. The ELPPRC is obviously \( \mathcal{NP} \)-complete since it can reduce to an ESPPRC by setting \( a = 0 \) and \( b = 1 \) in cost function \( f(w) \). This implies that optimally solving the pricing subproblem is computationally expensive. In the next section, we design an ad hoc label-setting algorithm to optimally solve the pricing subproblem.

### 3.4 Column Generation

Column generation is applied to solve the LMP (i.e., the linear relaxation of the formulation (3.11) – (3.17)) augmented by appropriate branching decisions and some cutting planes. For an overview of column generation, the reader is referred
to [11, 33]. The optimal solution value of the LMP is a lower bound of its associated branch-and-bound node. The column generation procedure cannot directly solve the LMP due to its inability of enumerating all variables $\theta_{r,p}$. Instead, it is an iterative procedure that alternates between solving a restricted linear relaxation of the master problem (RLMP) and a pricing subproblem. The RLMP is the LMP restricted to a subset of all variables $\theta_{r,p}$, which can be optimally solved by the simplex algorithm. The goal of solving the pricing subproblem is to identify the columns that have negative reduced costs with respect to the dual optimal solution of the current RLMP. If no such column is found, the column generation procedure is terminated with an optimal solution to the current RLMP, which is also an optimal solution to the LMP. Otherwise, we introduce one or more columns with negative costs into the current RLMP and restart the column generation iteration.

In this section, we first prove that the optimal solution of the pricing subproblem must be an extreme collection pattern. Based on this finding, we then develop a label-setting algorithm to solve the pricing subproblem. Finally, several strategies are introduced to accelerate the label-setting algorithm.

### 3.4.1 Extreme Collection Pattern

We first give the definition of the extreme collection pattern as follows.

**Definition 1** A collection pattern $p$ is an extreme collection pattern if and only if it is composed of zero collections ($\delta_i = 0$), full collections ($\delta_i = d_i$) and at most one split collection ($0 < \delta_i < d_i$).

Then, we can prove the following theorems.

**Theorem 3.2** Given a route $r$, any collection pattern $p \in P_r$ can be represented by a convex combination of extreme collection patterns in $P_r$. 
Proof: We assume $r = (v(1), v(2), \ldots, v(|r|))$, where $v(i)$ is the index of the $i$-th vertex, $v(1) = 0$ and $|r| \geq 2$ is the number of vertices in route $r$. Let $p = (\delta_{v(1)}, \delta_{v(2)}, \ldots, \delta_{v(|r|)})$ be an arbitrary feasible collection pattern compatible with route $r$. Then, $P_r$ is the feasible region defined by the following $|r| + 1$ constraints:

$$\sum_{i=1}^{|r|} \delta_{v(i)} \leq Q \quad (3.28)$$
$$0 \leq \delta_{v(i)} \leq d_{v(i)}, \forall 1 \leq i \leq |r| \quad (3.29)$$

It is easy to observe that $P_r$ is a closed convex set. Thus, any point in $P_r$ can be represented by a convex combination of the extreme points of $P_r$, each corresponding to an extreme collection pattern. There are $|r|$ decision variables $\delta_i$ ($1 \leq i \leq |r|$), so we must use $|r|$ active independent constraints to define each extreme point of $P_r$. In other words, only one of $|r| + 1$ independent constraints can be loose in an extreme point of $P_r$.

As a result, if one extreme point has a loose constraint, e.g., $0 < \delta_{v(k)} < d_{v(k)}$, it must have $\sum_{i=1}^{|r|} \delta_{v(i)} = Q$, and $\delta_{v(i)}$ equals either $d_{v(i)}$ or zero for all $v(i) \in r$ except $i = k$. If $\sum_{i=1}^{|r|} \delta_{v(i)} < Q$, the corresponding extreme point must have either $\delta_{v(i)} = d_{v(i)}$ or $\delta_{v(i)} = 0$ for all $v(i) \in r$. □

Theorem 3.3 One of the optimal solutions to the pricing subproblem must be an extreme collection pattern.

Proof: Assume the optimal solution of the pricing subproblem is a collection pattern compatible with route $r$. If route $r$, namely all variables $x_{i,j}$, is fixed, the pricing subproblem can be written as a linear relaxation of a bounded
knapsack problem:

\[
C(r, Q) = \min \sum_{i=1}^{\vert r \vert-1} c_{v(i),v(i+1)} \left( a \sum_{j=1}^{i} \delta_{v(j)} + b \right) \\
- \sum_{i=1}^{\vert r \vert} \left( \delta_{v(i)} \pi_{v(i)} + \mu_{v(i)} \right) 
\]

s.t. Constraints (3.28) and (3.29).

From this model, we can easily find that one of the optimal solutions to the pricing subproblem must be an extreme point of \( P_r \), which represents an extreme collection pattern. □

According to Theorem 3.3, we can solve the pricing subproblem to optimality by only examining all extreme collection patterns, which significantly reduces the search space of the label-setting algorithm. The objective function (3.30) can be rewritten as:

\[
f_r - \sum_{i=1}^{\vert r \vert} \delta_{v(i)} g_{v(i)} \tag{3.31}
\]

where

\[
f_r = \sum_{i=1}^{\vert r \vert-1} b c_{v(i),v(i+1)} - \sum_{i=1}^{\vert r \vert} \mu_{v(i)},
\]

\[
g_{v(i)} = \pi_{v(i)} - a \sum_{j=i}^{\vert r \vert-1} c_{v(j),v(j+1)}, \quad \forall \ 1 \leq i \leq \vert r \vert - 1,
\]

\[
g_{v(\vert r \vert)} = \pi_{v(\vert r \vert)}.
\]

This shows that the reduced cost of a collection pattern consists of two components: the first component \( f_r \) (called fixed cost) is a constant only related to route \( r \), while the second component \(- \sum_{i=1}^{\vert r \vert} \delta_{v(i)} g_{v(i)}\) (called variable cost) is determined by both route \( r \) and quantity \( \delta_{v(i)} \). The value of \( g_{v(i)} \) can be viewed as
the profit per unit product collected from vertex \( v(i) \).

Given a (partial) route \( r \) and the total collected quantity \( \hat{q} \) \((0 \leq \hat{q} \leq \sum_{i \in r} d_i)\) along this route, the minimal reduced cost \( G(r, \hat{q}) \) of all possible collection patterns can be computed using a greedy procedure shown in Algorithm 1. The collection pattern associated with \( G(r, \hat{q}) \) produced by the greedy procedure is obviously an extreme collection pattern. Note that when \( \hat{q} > \sum_{i \in r} d_i \), there does not exist feasible collection patterns. When route \( r \) is fixed, we can view \( G(r, q) \) as a function of \( q \), called the reduced cost function. From Algorithm 1, we observe that \( G(r, q) \) is a convex and continuous piece-wise linear function of \( q \). We illustrate this function in Figure 3.3, where \( G(r, 0) = f_r \), the slope \( s_{lk} = g_{v'(k)} \) and \( q_k = \sum_{i = 1}^{k} d_{v'(k)} \). We can directly get the value of \( C(r, Q) \) from \( G(r, q) \) by: \( C(r, Q) = G(r, q^*) = \min_{0 \leq q \leq Q} \{ G(r, q) \} \), which implies that \( \delta_{v(i)} \) is set to zero if \( g_{v(i)} \leq 0 \). Actually, to compute \( C(r, Q) \), we can perform a modified Algorithm 1 in which \( \hat{q} = Q \) is defined as the allowable capacity and only the vertices with \( g_{v(i)} > 0 \) are considered.

Algorithm 1 The greedy procedure of computing \( G(r, \hat{q}) \).

1: INPUT: \( g_{v(i)} \) and the total collected quantity \( \hat{q} \);
2: Set \( \delta_{v(i)} = 0 \) for all \( 1 \leq i \leq |r| \);
3: Sort all vertices in route \( r \) by decreasing value of \( g_{v(i)} \), yielding a sorted vertex list \((v'(1), v'(2), \ldots, v'(|r|))\).
4: The remaining capacity \( rc \leftarrow \hat{q} \);
5: \( k \leftarrow 1 \);
6: while \( rc \geq 0 \) and \( k \leq |r| \) do
7: \( \text{Set } \delta_{v'(k)} \leftarrow \min \{ d_{v'(k)}, rc \}, rc \leftarrow rc - \delta_{v'(k)} \) and \( k \leftarrow k + 1 \);
8: end while
9: Compute \( G(r, \hat{q}) \) according to Expression (3.31).

3.4.2 The Label-Setting Algorithm

The label-setting algorithm is a widely used technique to solve the pricing subproblems of various vehicle routing models, such the ESPPRC [49, 110], the
shortest path problem with resource constraints (SPPRC) [70] and the shortest path problem with time windows and linear node costs [69]. The aim of solving the pricing subproblem is to identify the complete routes with negative reduced cost, namely $C(r,Q)$. In our label-setting algorithm, a multi-dimensional label $E_i = (\tau_i, N_i, V_i, G(r,q) = (F_i, SL_i, I_i))$ is defined to represent a state associated with a feasible (partial) route $r$ from vertex 0 to vertex $i$, where:

- $\tau_i$ is the earliest service starting time at vertex $i$, which must lie within $[e_i, l_i]$;
- $N_i \subseteq V$ is the set of all visited vertices;
- $V_i \subseteq V$ is the set of all vertices that could be reached from route $r$;
- $G(r,q)$ is the reduced cost function associated with route $r$, which could be constructed using: $F_i = f_r$, $SL_i = \{g_j\}_{j \in r}$ and $I_i = \{d_j\}_{j \in r}$.

Typically, a label has one component indicating the (reduced) cost of the route and several other components recording the consumed resources, i.e., each component has a fixed value. However, our proposed label has a special component $G(r,q)$, namely a function of $q$. Additionally, in our label we do not
need a component related to the vehicle capacity. At vertex 0, we define $E_0 = (\tau_0, N_0, V_0, G(r, q) = (F_0, SL_0, I_0)) = (0, \{0\}, V_C \cup \{n + 1\}, (0, \emptyset, \emptyset))$. Each vertex may have multiple labels and the optimal solution to the pricing subproblem can be achieved by identifying the labels with the smallest $C(r, Q)$ at vertex $n + 1$.

A label $E_i$ can be extended to vertex $j \in V^+(i)$, yielding a new label $E_j$. The extension functions are:

- $\tau_j = \max\{e_j, \tau_i + s_i + t_{i,j}\}$;
- $N_j = N_i \cup \{j\}$;
- $V_j = V_i - \{k \in V^+(j) : \tau_j + s_j + t_{j,k} > l_k\} - \{j\}$;
- $F_j = F_i + bc_{i,j} - \mu_j$;
- $SL_j = \{g_k \leftarrow g_k - ac_{i,j} : g_k \in SL_i\} \cup \{g_j = \pi_j\}$;
- $I_j = I_i \cup \{d_j\}$.

Note that all labels do not contain any information regarding the order in which the vertices have been visited, and the labels $E_j$ with $V_j = \emptyset$ or $\tau_j > l_j$ are discarded. In the course of the label-setting algorithm, we cyclically examine all vertices, at each of which all labels that do not have successors would be extended. Extending a label at vertex $i$ may create as many new labels as the number of its successors. Undoubtedly, the number of labels would increase exponentially with the extension of the labels. To avoid exhaustive enumeration, dominance rules are employed to identify and eliminate the dominated labels. The performance of the label-setting algorithm heavily depends on the efficiency of the dominance rules, which determine the number of states generated.

The label $E_i$ can be regarded as a set of infinite number of labels $E_i(\hat{q}) = (\tau_i, N_i, V_i, \hat{q}, G(r, \hat{q}))$ for all $0 \leq \hat{q} \leq \sum_{i \in r} d_i$, where the quantity of products collected along the partial route is exactly $\hat{q}$ and $G(r, \hat{q})$ is the associated minimal
reduced cost. Let $p_i$ be the partial extreme collection pattern associated with $G(r, \hat{q})$ and $\bar{P}_i$ be the set of all feasible extensions of the label $E_i(\hat{q})$ to vertex $n + 1$. We use $p_i \oplus p'$ to denote the complete collection pattern resulting from extending $p_i$ by $p' \in \bar{P}_i$. As stated by Irnich and Desaulniers [70], Desaulniers [39], Dabia et al. [34], a label $E_i^1(\hat{q}^1)$ is dominated by a label $E_i^2(\hat{q}^2)$ if the following conditions hold:

- Any feasible extension of $E_i^1(\hat{q}^1)$ is also feasible to $E_i^2(\hat{q}^2)$, namely $\bar{P}_i^1 \subseteq \bar{P}_i^2$;
- The reduced cost of $p_i^1 \oplus p'$ is greater than or equal to that of $p_i^2 \oplus p'$ for each $p' \in \bar{P}_i^1$.

However, it is not straightforward to verify the above conditions since it requires to evaluate all feasible extensions of both labels. Instead, we propose the following sufficient conditions: $E_i^1(\hat{q}^1)$ is dominated by $E_i^2(\hat{q}^2)$ if

1. $\tau_i^2 \leq \tau_i^1$;
2. $V_i^2 \supseteq V_i^1$;
3. $\hat{q}^2 \leq \hat{q}^1$
4. $G(r^2, \hat{q}^2) \leq G(r^1, \hat{q}^1)$

The dominance rule for two labels $E_i(\hat{q}^1)$ and $E_i(\hat{q}^2)$ that have the same route $r$ can be described as: $E_i(\hat{q}^1)$ is dominated by $E_i(\hat{q}^2)$ if conditions 3 and 4 are satisfied. Based on this dominance rule, the labels $E_i(\hat{q})$ associated with the increasing part of the reduced cost function $G(r, q)$ (see Figure 3.3) are dominated by label $E_i(q^*)$ and can be safely eliminated. Therefore, we can replace the increasing part of the reduced cost function with a zero slope piece and redefine $G(r, \hat{q})$ as the minimal reduced cost associated with route $r$ and an allowable capacity $\hat{q}$. Subsequently, we derive a dominance rule for two labels $E_i^1$ and $E_i^2$ as:
$E_1^i$ is dominated by $E_2^i$ if a label $E_2^i(\hat{q}^2)$ can be always found to dominate $E_1^i(\hat{q}^1)$ for each feasible $\hat{q}^1$. Specifically, the sufficient conditions for $E_2^i$ to dominate $E_1^i$ are: conditions 1 and 2 are satisfied, and $G^1(r^1, q) \geq G^2(r^2, q)$ for each $q \in [0, Q]$ (see Figure 3.4). Since this dominance rule involves only two labels, we call it the pair dominance rule.

![Figure 3.4: The graphic representations of the functions $G^1(r^1, q)$ and $G^2(r^2, q)$ (note that the increasing parts of both functions have been replaced with zero slope pieces).](image)

The pair dominance rule is quite weak since the number of cases that a reduced cost function lies below another one may not be very large. Fortunately, we find that although a label cannot be dominated by another one, it might be dominated by a set of labels. Based on this finding, we introduce a novel dominance rule called the set dominance rule, which is described as follows. It is easy to derive that $E_1^i$ is a dominated label if a label $E_x^i(\hat{q}^x)$ can always be found to dominate $E_1^i(\hat{q}^1)$ for each feasible $\hat{q}^1$. Denoting by $E_i$ the set of all labels ending at vertex $i$, we can define a label set $E_i^1$ related to label $E_1^i$ as: $E_i^1 = \{ E_x^i \in E_i : \tau_x^i \leq \tau_i^1, V_x^i \supseteq V_i^1, E_x^i \neq E_1^i \}$. Using the labels in $E_i^1$, we can construct a minimal reduced cost function:

$$G_{\text{min}}^1(q) = \min_{E_i^1 \in E_i^1} \{ G_x^i(r^x, q) \} \quad (3.32)$$
As illustrated in Figure 3.5, the function $G_{\text{min}}^1(q)$ is composed of the minimal part of all functions $G^x(r^x, q)$ $(x = 2, 3, 4)$, and is not necessarily convex. If the curve $G_{\text{min}}^1(q)$ lies below the curve $G^1(r, q)$, we say that $E_i^1$ is dominated by the label set $E_i^1$ and can be safely discarded. Figure 3.6 gives an example in which $E_i^1$ cannot be dominated by either $E_i^2$ or $E_i^3$, but it is dominated by set $\{E_i^2, E_i^3\}$ according to the set dominance rule. The implementation details of our set dominance rule is described in Appendix A.

Figure 3.5: (a) The graphic representations of functions $G^2(r^2, q)$, $G^3(r^3, q)$ and $G^4(r^4, q)$. (b) The graphic representation of function $G_{\text{min}}^1(q)$.

Figure 3.6: (a) The graphic representations of functions $G^1(r^1, q)$, $G^2(r^2, q)$ and $G^3(r^3, q)$. (b) $G_{\text{min}}^1(q)$ lies below $G^1(r^1, q)$. 
The label-setting algorithm proposed by Desaulniers [39] can be adapted to solve our pricing subproblem. However, compared with that algorithm, our proposed label-setting algorithm exhibits obvious advantages in the following two aspects. First, in the course of our label-setting algorithm, each feasible partial route has at most one label. The label-setting algorithm in Desaulniers [39] creates a huge number of dominated labels that cannot be efficiently eliminated by their proposed pair dominance rule. When extending a label from vertex $i$ to vertex $j$ ($j \neq n + 1$), their label-setting algorithm creates up to three labels, corresponding to zero delivery, full delivery and split delivery, respectively. This type of extension is essentially equivalent to enumerating all feasible extreme delivery patterns compatible with a certain route and as a result a feasible partial route is very likely to be associated with a large number of labels. However, in fact, for each feasible partial route, only at most one label is non-dominated and needs to be kept. Second, our proposed set dominance rule is far more efficient to eliminate the dominated labels since it utilizes the information of all available labels ending at a certain vertex. In brief, the label-setting algorithm proposed by Desaulniers [39] creates and deals with much more labels than ours, and consequently requires more computational efforts. To show the superiority of our label-setting algorithm, we applied it to solve the LMP for all SDVRPTW instances used in Desaulniers [39], and report the experimental results in Section 4.5.

### 3.4.3 Implementation Details of Set Dominance Rule

In this section, we address the implementation details of the set domination rule. We eliminated the dominated labels ending at vertex $i$ by maintaining a directed dominance graph $G_i = (N_i, A_i)$. For the rest of this section, we distinguish between the terms vertex and node, which are usually considered the same and are
used interchangeably; we specify that vertex refers to the vertex in the underlying graph \( G \) of the SCVRPTWL, and node refers to the node in the dominance graph \( G_i \).

Each node \( u \in N_i \) includes a set \( L_u \) of non-dominated labels that end at vertex \( i \) with the same \( \tau_i \) and \( V_i \) and has three attributes: the earliest service starting time \( \tau_i(L_u) \), the set \( V_i(L_u) \) of reachable vertices and the minimum reduced cost function \( G_{\min}(L_u, q) = \min_{E^r \in L_u} \{G^r(r^x, q)\} \) for \( q \in [0, Q] \). The directed edge \((u, v)\) is included in the edge set \( E_i \) if there does not exist other paths from node \( u \) to node \( v \) and at least one of the following conditions holds:

1. \( \tau_i(L_u) \leq \tau_i(L_v) \) and \( V_i(L_u) \supset V_i(L_v) \);
2. \( \tau_i(L_u) < \tau_i(L_v) \) and \( V_i(L_u) \supseteq V_i(L_v) \).

We update \( G_{\min}(L_v, q) = \min\{G_{\min}(L_u, q), G_{\min}(L_v, q)\} \) with the creation of edge \((u, v)\). In this dominance graph, there must exist a root node 0 (a node that does not have incoming edges), which corresponds to the two-vertex partial route \( r = (0, i) \). An example of the dominance graph is given in Figure 3.7. Starting from the root node 0, we update \( G_i \) by invoking Algorithm 2 every time a new label ending at vertex \( i \) is created. After performing this algorithm, the labels included in \( G_i \) are all currently non-dominated.

![Figure 3.7: An example of the dominance graph \( G_i \).](image-url)
Algorithm 2 The Process of Updating Graph $G_i$. 

1: INPUT: the current graph $G_i$; 
2: Check whether label $E_i^x$ is dominated; 
3: if $E_i^x$ is not dominated then 
4: Insert $E_i^x$ into the label set of a certain existing node or create a new node whose label set contains only $E_i^x$; 
5: Remove from $G_i$ the previously inserted labels that become dominated after the adding of $E_i^x$; 
6: else 
7: Discard $E_i^x$. 
8: end if

We check whether a label $E_i^x = (\tau_i^x, N_i^x, V_i^x, G_i^x(r, q))$ is dominated (line 2 in Algorithm 2) by performing a recursive procedure $DominanceCheck(u, E_i^x, G_i)$ shown in Algorithm 3. Given a node $u \in G_i$, if there exists a child $v$ of $u$ that satisfies condition 1 or 2 (or both), the procedure moves to node $v$ since $G_{\min}(L_v, q)$ has more chance to lie below $G_i^x(r, q)$. Otherwise, the procedure checks whether $E_i^x$ is a dominated label, i.e., whether $G_{\min}(L_u, q) \leq G_i^x(r, q)$ holds for each $q \in [0, Q]$.

Algorithm 3 $DominanceCheck(u, E_i^x, G_i).$

1: $flag \leftarrow$ false; 
2: for each child $v$ of $u$ that has not been examined do 
3: if $\tau_i(L_v) \leq \tau_i^x, V_i(L_v) \supset V_i^x$ or $\tau_i(L_v) < \tau_i^x, V_i(L_v) \supseteq V_i^x$ then 
4: $flag \leftarrow$ true; 
5: $result \leftarrow DominanceCheck(v, E_i^x, G_i)$; 
6: if $result = true$ then 
7: return true; 
8: end if 
9: end if 
10: end for 
11: if $flag = false$ then 
12: if $G_{\min}(L_u, q) \leq G_i^x(r, q)$ for each $q \in [0, Q]$ then 
13: return true; 
14: else 
15: return false; 
16: end if 
17: end if 
18: return false.

If $E_i^x$ is a dominated label, we discard it (line 7 in Algorithm 2); otherwise,
we need to add it in \( \mathcal{G}_i \). We either insert the non-dominated \( E_i^x \) into \( L_u \) if \( \tau_i(L_u) = \tau_i^x \) and \( V_i(L_u) = V_i^x \) or create a new node whose label set contains only \( E_i^x \) if no such \( u \) exists (line 4 in Algorithm 2). The function \( \text{SearchNode}(u, E_i^x, \mathcal{G}_i) \) presented in Algorithm 4 is used to check whether \( \mathcal{G}_i \) contains a node \( u \) with \( \tau_i(L_u) = \tau_i^x \) and \( V_i(L_u) = V_i^x \). After inserting \( E_i^x \) into \( L_u \), we update \( G_{\min}(L_u, q) = \min\{G_{\min}(L_u, q), G^x(v^x_q, q)\} \) for all \( q \in [0, Q] \). The new node is created and connected to \( \mathcal{G}_i \) by invoking the function \( \text{CreateNode}(u, E_i^x, \mathcal{G}_i) \) shown in Algorithm 5. In this algorithm, we first create a node \( w \) and then connect it to \( \mathcal{G}_i \). If a node \( u \) satisfies \( \tau_i(L_u) \leq \tau_i(L_w) \) and \( V_i(L_u) \supseteq V_i(L_w) \), and none of its child has this relationship, we create edge \((u, w)\). Then, for each child \( v \) of \( u \), if \( \tau_i(L_v) \geq \tau_i(L_w) \) and \( V_i(L_v) \subseteq V_i(L_w) \), we remove edge \((u, v)\) and create edge \((w, v)\). The newly created node \( w \) may have multiple immediate predecessors and successors.

\begin{algorithm}
\caption{SearchNode\((u, E_i^x, \mathcal{G}_i)\).}
1: \textbf{if} \( \tau_i(L_u) = \tau_i^x \) and \( V_i(L_u) = V_i^x \) \textbf{then}
2: \hspace{1em} \textbf{return} \( u \);
3: \textbf{end if}
4: \hspace{1em} \textbf{for} Each child \( v \) of \( u \) that has not been examined \textbf{do}
5: \hspace{2em} \textbf{if} \( \tau_i(L_v) \leq \tau_i^x \) and \( V_i(L_v) \supseteq V_i^x \) \textbf{then}
6: \hspace{3em} \textbf{return} \( \text{SearchNode}(v, E_i^x, \mathcal{G}_i) \);
7: \hspace{2em} \textbf{end if}
8: \hspace{1em} \textbf{end for}
9: \hspace{1em} \textbf{return} \( \text{null} \).
\end{algorithm}

The adding of label \( E_i^x \) in graph \( \mathcal{G}_i \) necessitates the improvement of the minimal reduced cost function at all successor nodes. Moreover, some labels in \( \mathcal{G}_i \) may become dominated due to the adjustment of the minimal reduced cost function and therefore can be removed (line 5 in Algorithm 2). To improve the minimal reduced cost function and remove the dominated labels, we invoke a function \( \text{RemoveLabel}(u, \mathcal{G}_i) \), which is shown in Algorithm 6.
Algorithm 5 CreateNode($u$, $E_i^x$, $G_i$).

1: $\text{flag} \leftarrow \text{false};$
2: Create a node $w$ with $L_w = \{E_i^x\}$, $\tau_i(L_w) = \tau_i^x$, $V_i(L_w) = V_i^x$ and $G_{\text{min}}(L_w, q) = G^x(r^x, q);$  
3: for each child $v$ of $u$ that has not been examined do  
4: if $\tau_i(L_v) \leq \tau_i^x$ and $V_i(L_v) \supseteq V_i^x$ then  
5: $\text{flag} \leftarrow \text{true};$
6: CreateNode($v$, $E_i^x$, $G_i$);  
7: end if  
8: end for  
9: if $\text{flag} = \text{false}$ then  
10: Create edge $(u, w)$ and update $G_{\text{min}}(L_w, q) = \min\{G_{\text{min}}(L_u, q), G_{\text{min}}(L_w, q)\}$ for all $q \in [0, Q]$;  
11: for each child $v$ of $u$ do  
12: if $\tau_i(L_v) \geq \tau_i(L_w)$ and $V_i(L_v) \subseteq V_i(L_w)$ then  
13: Remove edge $(u, v)$ and create edge $(w, v)$.  
14: end if  
15: end for  
16: end if

3.4.4 Accelerating Strategies

We have implemented the following three techniques to speed up the column generation procedure.

3.4.4.1 Bounded Bidirectional Search

As discussed in Righini and Salani [109, 110], the label-setting algorithm can be accelerated by bounded bidirectional search strategy. The resulting algorithm is called the bounded bidirectional label-setting (BBLS) algorithm whose procedure can be briefly summarized as the following three steps: (1) labels are extended forward from vertex 0, generating a set of forward partial routes; (2) labels are extended backward from vertex $n + 1$, generating a set of backward partial routes; and (3) pairs of forward and backward partial routes are joined together to generate complete routes.

The BBLS algorithms have been successfully employed to solve the pricing subproblems for a variety of vehicle routing models, such as the VRPTW [42], the
Algorithm 6 RemoveLabel($u, G_i$).

1: flag ← false;
2: for each child $v$ of $u$ that has not been examined do
3:     for each label $E^{r^x}_i \in L_v$ do
4:         if $G_{\text{min}}(L_u, q) \leq G^r(r^x, q)$ for all $q \in [0, Q]$ then
5:             Remove $E^{r^x}_i$ from $L_v$;
6:         end if
7:     end for
8:     if $L_v = \emptyset$ then
9:         Create an edge from $u$ to each child of $v$;
10:        Remove $v$ from $G_i$;
11:    else
12:        $G_{\text{min}}(L_v, q) = \min G_{\text{min}}(L_u, q), G_{\text{min}}(L_v, q)$;
13:    end if
14: end for

VRP with simultaneous distribution and collection [37], the pickup and delivery problem with time windows [112], the SDVRPTW [39] and the VRPTW with multiple use of vehicles [11]. In the BBLS algorithms developed in the above-mentioned articles, the forward and backward extensions are almost the same due to the symmetric structure of the investigated problems. We find that a common objective of these problems is to minimize the overall traveling distance of all vehicles. This feature results in a property that the costs of backward and forward partial routes are independent, i.e., the cost of a backward partial route does not rely on its forward partial route.

If the route cost is determined by the arrival time at each vertex or the flow on each edge, the symmetric structure of the vehicle routing models would be destroyed to a certain extent; we call this type of cost the *cumulative cost*. In the vehicle routing models with cumulative costs, e.g., the SCVRPTWL, we can find that the cost of a backward partial route is heavily relied on its forward partial route. In Ribeiro et al. [107], the authors developed a branch-price-and-cut algorithm to solve the workover rig routing problem (WRRP) that incorporates a cumulative cost at each vertex. They claimed that the bounded bidirectional search strategy cannot be applied to their pricing subproblem. We could not
make a conclusion on whether there exists a BBLS algorithm for the WRRP. However, after carefully analyzing the structure of our pricing subproblem, we find that it can still be optimally solved by a tailored BBLS algorithm in which the forward and backward extensions are considerably different.

The process of the bidirectional search strategy is pictorially shown in Figure 3.8. Given a backward partial route \( r = (v(1), v(2), \ldots, v(|r|)) \), where \(|r| \geq 2\) and \(v(|r|) = n + 1\), and the incoming flow \( \hat{q} \), the minimal reduced cost \( G^b(r, Q - \hat{q}) \) of all backward partial collection patterns can be computed by:

\[
G^b(r, Q - \hat{q}) = f^b_r - \sum_{i=1}^{|r|} \delta_{v(i)}g^b_{v(i)} \tag{3.33}
\]

\[
s.t. \quad \sum_{i=1}^{|r|} \delta_{v(i)} = Q - \hat{q} \\
0 \leq \delta_{v(i)} \leq d_{v(i)}, \forall 1 \leq i \leq |r| \]

where

\[
f^b_r = \sum_{i=1}^{|r|-1} \left( (aQ + b)c_{v(i),v(i+1)} - \mu_{v(i)} \right); \\
g^b_{v(1)} = \pi_{v(1)}; \\
g^b_{v(i)} = a \sum_{j=1}^{i-1} c_{v(j),v(j+1)} + \pi_{v(i)}, \forall 2 \leq i \leq |r| - 1; \\
g^b_{v(|r|)} = a \sum_{j=1}^{|r|-1} c_{v(j),v(j+1)}.
\]

We show the detailed derivation of \( G^b(r, Q - \hat{q}) \). Given a feasible partial backward route \( r = (v(1), v(2), \ldots, v(|r|)) \), where \(|r| \geq 2\) and \(v(|r|) = n + 1\), and the incoming flow \( \hat{q} \), the reduced cost \( G^b(r, Q - \hat{q}) \) can be computed by the
following model:

\[
G^b(r, Q - \hat{q}) = \min \sum_{i=1}^{r-1} c_{v(i),v(i+1)} \left( a(\hat{q} + \sum_{j=1}^{i} \delta_{v(j)}) + b \right) \\
- \sum_{i=1}^{r} \left( \delta_{v(i)} \pi_{v(i)} + \mu_{v(i)} \right) \\
\text{s.t.} \quad \sum_{i=1}^{r} \delta_{v(i)} = Q - \hat{q} \\
0 \leq \delta_{v(i)} \leq d_{v(i)}, \quad \forall \ 1 \leq i \leq |r|
\]  

(3.34)  

(3.35)  

(3.36)
The objective can be rewritten as:

\[
G^b(r, Q - \hat{q}) = a \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} + a \delta_v(i) \sum_{j=i}^{|r|-1} c_{v(j),v(j+1)} \\
+ b \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} - \sum_{i=1}^{|r|} \left( \delta_v(i) \pi_v(i) + \mu_v(i) \right) \\
= a \left( Q - \sum_{i=1}^{|r|} \delta_v(i) \right) \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} + a \delta_v(i) \sum_{j=i}^{|r|-1} c_{v(j),v(j+1)} \\
+ b \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} - \sum_{i=1}^{|r|} \left( \delta_v(i) \pi_v(i) + \mu_v(i) \right) \\
= a Q \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} + a \delta_v(i) \sum_{j=i}^{|r|-1} c_{v(j),v(j+1)} - \sum_{i=1}^{|r|-1} \mu_v(i) \\
+ a \delta_v(i) \sum_{j=i}^{|r|-1} c_{v(j),v(j+1)} - \pi_v(i) \delta_v(i) \\
= a Q \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} + b \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} \\
- \sum_{i=1}^{|r|-1} \mu_v(i) - \delta_v([r]) a \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} \\
- \sum_{i=1}^{|r|-1} \delta_v(i) \left( a \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} + \pi_v(i) - a \sum_{j=i}^{|r|-1} c_{v(j),v(j+1)} \right) \\
= \sum_{i=1}^{|r|-1} \left( (a Q + b) c_{v(i),v(i+1)} - \mu_v(i) \right) - \delta_v([r]) a \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} \\
- \sum_{i=1}^{|r|-1} \delta_v(i) \left( a \sum_{i=1}^{|r|-1} c_{v(i),v(i+1)} + \pi_v(i) - a \sum_{j=i}^{|r|-1} c_{v(j),v(j+1)} \right) \\
(3.37)
\]

Algorithm 1 can still be used to compute the value of \(G^b(r, Q - \hat{q})\). When route \(r\) is fixed, \(G^b(r, Q - q)\) can be viewed as a function of the allowable capacity \(Q - q\) (e.g., see Figure 3.9).

We use a label \(E^b_i = (\tau^b_i, N^b_i, V^b_i, G^b(r, Q - q) = (F^b_i, SL^b_i, I^b_i))\) to represent a
Figure 3.8: The bidirectional search strategy.

Figure 3.9: Graphic representation of the reduced cost function $G^b(r, Q - q)$.

state in backward extension, where:

- $\tau^b_i$ represents the minimum time which must be consumed since the departure from vertex $i$ up to the arrival at vertex $n + 1$;
- $N^b_i$ and $V^b_i$ have the same definitions as $N_i$ and $V_i$;
- $G^b(r, Q - q)$ represents the reduced cost function associated with the backward partial route $r$ and the allowable capacity $Q - q$, which could be represented by: $F^b_i = f^b_r$, $SL^b_i = \{g^b_j\}_{j \in r}$ and $I^b_i = \{d_j\}_{j \in r}$.

We define $E^b_{n+1} = (\tau^b_{n+1}, N^b_{n+1}, V^b_{n+1}, G^b(r, Q - q) = (F^b_{n+1}, SL^b_{n+1}, I^b_{n+1}), \{0, n + 1\}, V_C \cup \{0\}, (0, \{g^b_{n+1}\}, \{d_{n+1}\}))$. The overall time resource $T$ is equal to the maximum feasible arrival time at vertex $n + 1$, namely $T = \max_{i \in V_C} \{l_i +$
\(s_i + t_{i,n+1}\). Analogously, the backward partial route is extended from vertex \(j\) to vertex \(i\) according to the following functions:

- \(\tau_i^b = \max\{T - l_i - s_i, \tau_j^b + s_j + t_{i,j}\}\);
- \(N_i^b = N_j^b \cup \{i\}\);
- \(V_i^b = V_j^b - \{k \in V^-(i) : \tau_i^b + s_i + t_{k,i} > T - e_k - s_k\} - \{i\}\);
- \(F_i^b = F_j^b + (aQ + b)c_{i,j} - \mu_i\);
- \(SL_i^b = \{g_k \leftarrow g_k + ac_{i,j} : g_k \in SL_j^b\} \cup \{g_i = \pi_i\}\);
- \(I_i^b = I_i^b \cup \{d_i\}\).

We discard the labels with \(V_i^b = \emptyset\) or \(\tau_i > T - e_i - s_i\), and still use the set dominance rule that is described in Section 3.4.2 to eliminate the dominated backward partial routes.

When applying the BBLS algorithm, we consider time as the critical resource and only extend forward and backward labels whose consumed time resources are less than \(T/2\), namely \(\tau_i < T/2\) and \(\tau_i^b < T/2\). A forward label \(E_i = (\tau_i, N_i, V_i, G(r,q))\) and a backward label \(E_j^b = (\tau_j^b, N_j^b, V_j^b, G^b(r,Q-q))\) can be joined together to form a complete feasible route if \(\tau_i + s_i + t_{i,j} + \tau_j^b \leq T\) and \(N_i \cap N_j^b = \emptyset\). The cost of the resulting complete collection pattern is achieved using the information of \(G(r,q)\) and \(G^b(r,Q-q)\) as follows. The fixed cost of the complete collection pattern is the sum of \(F_i + F_j^b + bc_{i,j}\). With the values of \(g_k \in SL_i \cup SL_j^b\) and the collected quantity \(Q\), we can use Algorithm 1 to decide the quantity \(\delta_k\) collected at each visited vertex \(k\) and then compute the variable cost of the complete collection pattern as \(-\sum_{k \in N_i \cup N_j^b} \delta_k g_k\). The minimum cost among all complete collection patterns is the optimal solution value of the pricing subproblem. Usually, at each column generation iteration we identify a number of columns with negative reduced cost and then add them into the current RLMP.
3.4.4.2 Heuristic Column Generator

Heuristics may identify negative reduced cost columns with much less computation time, compared to the exact label-setting algorithm. To avoid solving the pricing subproblem optimally at each column generation iteration, we develop an adaptive greedy heuristic (AGH), as shown in Algorithm 7, to heuristically and rapidly identify negative reduced cost columns. At each column generation iteration, we first use the AGH to solve the pricing subproblem. If it manages to obtain some columns with negative reduced cost, we add these columns into the RLMP and start the next iteration. Otherwise, we invoke the BBLs algorithm to solve the pricing subproblem to optimality.

The AGH tries to identify up to $\text{maxCol}$ negative reduced cost columns, starting with a set $R_0$ of routes with zero reduced cost in the optimal solution of the current RLMP. We define $\text{maxIter}$ as the maximum number of iterations associated with each route in $R_0$, $\rho_i$ as a valuation for each vertex $i$ that is used to calculate its priority value, and $\eta$ ($0 < \eta < 1$) as a penalty factor. Since the best extreme collection pattern of a given route can be easily obtained, in the AGH we use a route to represent a solution of the pricing subproblem. At the beginning of the outer loop, the valuations of all vertices are initialized to one and all vertices that are not included in the current route $r$ are stored in a vertex queue $\text{vertex_queue}$ in an order of decreasing value of $\pi_i \times \rho_i$ (lines 8 – 9 in Algorithm 7).

In each iteration, the heuristic pops the vertices $u$ one by one from $\text{vertex_queue}$, checks their insertion into the current route $r$ by a subroutine $\text{GreedyInsert}(r, u)$ given in Algorithm 8 and adds the resulting feasible routes with negative reduced cost into a route pool $\text{route_pool}$ (see lines 11 – 27 in Algorithm 7). Upon collecting $\text{maxCol}$ negative reduced cost columns, we terminate the heuristic and return $\text{route_pool}$ (lines 19 – 21 in Algorithm 7). If the resulting route $r'$ is better
than the currently best route \( r^* \), we update \( r^* \) by \( r' \). Otherwise, the corresponding parameter \( \rho_u \) is decreased to \( \eta \times \rho_u \), which delays the checking of its insertion in the next iteration. This strategy is very similar to the adaptive process used by other search techniques such as the ejection pool algorithm \[89\]. Whenever vertex queue becomes empty, the heuristic removes one non-depot vertex \( u \) from the current route by alternatively using a greedy procedure GreedyRemove\((r)\) shown in Algorithm[9] and a random way, decrease \( \rho_u \) to \( \eta \times \rho_u \), and reset vertex queue based on toggling between two sorting rules (see lines 28 – 38 in Algorithm 7).

To obtain the columns, we compute the best collection patterns compatible with each route \( u \) in route pool by performing Algorithm 1 with consideration of only the vertices with \( g_v(i) > 0 \).

### Algorithm 7 The Adaptive Greedy Heuristic

1. INPUT: A set of routes \( R_0 \), maxCol, maxIter and \( \eta \);
2. Define vertex_queue and route_pool as a vertex queue and a route pool, respectively;
3. Set \( \text{flag} \leftarrow \text{false} \), \( k \leftarrow \text{maxIter} \) and \( r^* \leftarrow \text{any route } r \in R_0 \);
4. while \( k \leq \text{maxIter} \) and \( R_0 \) is not empty do
5.   if \( \text{flag} = \text{false} \) then
6.     \( k \leftarrow 1 \);
7.     \( r \leftarrow \text{randomly select one route from } R_0 \) and remove \( r \) from \( R_0 \);
8.     Set \( \rho_i \leftarrow 1 \) for all \( i \in V \) and vertex_queue \( \leftarrow V \cup \{ \text{all vertices in } r \} \);
9.     Sort all vertices \( i \) in vertex_queue by decreasing value of \( \pi_i \times \rho_i \);
10. end if
11. while vertex_queue is not empty do
12.    \( u \leftarrow \text{pop the top element in } \text{vertex_queue} \);
13.    \( r' \leftarrow \text{GreedyInsert}(r, u) \);
14.    if \( r' \neq \text{null} \) and \( C(r', Q) < C(r^*, Q) \) then
15.       \( r \leftarrow r' \);
16.    end if
17.    if \( r' \neq \text{null} \) and \( C(r', Q) \) is negative then
18.       Add \( r' \) into route_pool;
19.    end if
20.    if the size of route_pool is equal to maxCol then
21.       return route_pool;
22.    end if
23.    if \( C(r', Q) < C(r^*, Q) \) then
24.       \( r^* \leftarrow r' \);
25.       \( \rho_u \leftarrow \rho_u / \eta \);
26.    end if
27. end while
28. if \( \text{flag} = \text{false} \) then
29.    \( r \leftarrow \text{GreedyRemove}(r) \);
30.    vertex_queue \( \leftarrow V \cup \{ \text{all vertices in } r \} \);
31.    Sort all vertices \( i \) in vertex_queue by decreasing value of \( d_i \times \pi_i \times \rho_i \);
32.    \text{flag} \leftarrow \text{true};
33. else
34.    \( r \leftarrow \text{randomly remove a vertex } u \text{ except 0 and } n + 1 \text{ from } r \) and set \( \rho_u \leftarrow \eta \times \rho_u \);
35.    vertex_queue \( \leftarrow V \cup \{ \text{all vertices in } r \} \);
36.    Sort all vertices \( i \) in vertex_queue by decreasing value of \( \pi_i \times \rho_i \);
37.    \text{flag} \leftarrow \text{false};
38. end if
39. \( k \leftarrow k + 1 \);
40. end while
41. return route_pool;
Algorithm 8 GreedyInsert($r, i$).

1: $r' \leftarrow \text{null}$;
2: for each pair of two consecutive vertices $u$ and $v$ in $r$ do
3: 
4: Insert $i$ between $u$ and $v$;
5: if the resulting $r$ is infeasible then
6: Continue;
7: else if $r'$ is not initialized then
8: $r' \leftarrow r$;
9: else if $C(Q, r) < C(Q, r')$ then
10: $r' \leftarrow r$;
11: end if
12: Restore $r$ to its state before inserting $i$;
13: end for
14: return $r'$.

Algorithm 9 GreedyRemove($r$).

1: $r' \leftarrow \text{null}$;
2: for each vertex $u \in r$ except 0 and $n + 1$ do
3: Remove $u$ from $r$;
4: if $r'$ is not initialized then
5: $r' \leftarrow r$;
6: else if $C(Q, r) < C(Q, r')$ then
7: $r' \leftarrow r$ and $v \leftarrow u$;
8: end if
9: Restore $r$ to its state before deleting $u$;
10: end for
11: $\rho_v \leftarrow \eta \times \rho_v$;
12: return $r'$.

3.4.4.3 Decremental Search Space

The decremental search space was introduced independently by Boland et al. [26] and Righini and Salani [110]. It starts from solving the pricing subproblem with the elementary requirements of all customers being relaxed, i.e., each customer can be visited more than once in a route. In our label-setting algorithm, if the elementary requirement of vertex $j$ is relaxed, it will not be removed from $V_j$ when the label is extended from vertex $i$ to vertex $j$, and is allowed to exist in $N_i \cap N_j^b$ when joining labels. If the computed least-cost path is nonelementary, the customers that are visited more than once are required to be elementary and the pricing subproblem is solved again. This process is repeated until an
elementary least-cost route is found. Our implementation of the decremental search space technique is the same as the one described in Desaulniers [39]. This acceleration technique has also been employed in the branch-and-price algorithms for solving several other vehicle routing models, such as the VRP with discrete split deliveries and time windows [113], the VRP with deliveries, selective pickups and time windows [62] and the multi-depot VRPTW [22].

3.5 Branch-and-Price-and-Cut Algorithm

Branch-and-price-and-cut is one of the leading solution procedures for many large-scale integer programming models (e.g., see Ropke and Cordeau [112], Barnhart et al. [15], Belov and Scheithauer [18], Hwang et al. [67]). Over the course of the branch-and-bound search, some violated valid inequalities are dynamically added into the model. In our branch-and-price-and-cut algorithm, the initial set of columns corresponds to the set of all one-customer routes, namely $r = (0, i, n+1)$ for each $i \in V_C$. At each branch-and-bound node, we first optimally solve the LMP using the column generation procedure to obtain a lower bound. For the node that cannot be pruned, we next try to identify the $k$-path inequalities and strong minimum number of vehicles inequalities that are violated by the current linear solution. If such violated inequalities are found, we add them into the model and invoke the column generation procedure again to further improve the lower bound. The above procedure is repeated until the node is pruned or no violated inequalities can be found.

In this section, we first describe two types of valid inequalities. This is followed by search and branching strategies that guide the exploration of the branch-and-bound tree.
3.5.1 Valid Inequalities

We use two types of valid inequalities for the SCVRPTWL, namely the $k$-path inequality and the *strong minimum number of vehicles inequalities*, which have been implemented by Archetti et al. [8] for the SDVRPTW. These inequalities are defined on the master problem variables $\theta_{r,p}$. After adding some valid inequalities into the master problem, the subproblem as well as the label-setting algorithm need to be modified accordingly. Below, we only discuss in detail the treatment of these inequalities in forward extension. The modifications on backward extension can be easily derived in a similar manner.

3.5.1.1 $k$-path Inequalities

The $k$-path inequalities are expressed as:

$$
\sum_{r \in R} \sum_{p \in P_r} \sum_{(i,j) \in E^-(S)} \beta_{i,j,r} \theta_{r,p} \geq \left\lceil \sum_{i \in S} d_i \right\rceil \frac{Q}{K}, \quad \forall \ S \in \Gamma \tag{3.38}
$$

where the binary parameter $\beta_{i,j,r} = 1$ if edge $(i,j)$ is used in route $r$, $E^-(S) = \{(i, j) \in E | i \in S, j \notin S\}$ is the set of edges leaving the customer subset $S$, and $\Gamma$ is the set of the subsets $S \in V_C$. Let $\lambda = (\lambda_{S_1}, \ldots, \lambda_{S_{|\Gamma|}})$ be the values of the dual variables associated with Constraints (3.38). The reduced cost $\bar{c}_{r,p}$ and the fixed cost $f_r$ become:

$$
\bar{c}_{r,p} = c_{r,p} - \sum_{i \in r} (\delta_{i,p} \pi_i + \mu_i) - \sum_{S \in \Gamma} \sum_{(i,j) \in E^-(S) \cap r} \lambda_S
$$

$$
f_r = \sum_{i=1}^{r-1} \left( bc_{v(i),v(i+1)} - \sum_{S \in \Gamma(v(i),v(i+1)) \in E^-(S)} \lambda_S \right) - \sum_{i=1}^{|r|} \mu_{v(i)}
$$

Handling the new dual variable $\lambda_S$ in the label-setting algorithm needs to
modify the extensions function related to the fixed cost as follows:

\[ F_j = F_i + b c_{i,j} - \mu_j - \sum_{S \in \Gamma_{(i,j)} \in E^-(S)} \lambda_S \]

Moreover, when joining two labels, the fixed cost of the complete collection pattern becomes \( F_i + F_j^b + b c_{i,j} - \sum_{S \in \Gamma_{(i,j)} \in E^-(S)} \lambda_S \). It is worthy to mention that the introduction of the \( k \)-path inequalities only affects the fixed cost of the reduced cost.

To identify the violated \( k \)-path inequalities, we have implemented three types of separation heuristics, which have been used in the branch-and-price-and-cut algorithms for the SDVRPTW [39, 8]. The first one was the \textit{partial enumeration heuristic} proposed by Desaulniers [39] and the other two were the \textit{extended shrinking heuristic} and the \textit{route-based algorithm} developed by Archetti et al. [8]. We refer the reader to these two articles for full details of these three separation heuristics. Note that our \( k \)-path inequalities only take the vehicle capacity constraints into consideration.

### 3.5.1.2 Strong Minimum Number of Vehicles (SMV) Inequalities

Define \( V_S \) as the set of customers \( i \in V_C \) with \( d_i \leq Q \). The SMV inequalities are expressed as:

\[
\sum_{r \in R} \sum_{p \in P_r} (2\alpha_{i,r,p}^F + \alpha_{i,r,p}^{SZ}) \theta_{r,p} \geq 2. \forall \ i \in V_S
\]

(3.39)

where at customer \( i \) in pattern \( p \) compatible with route \( r \), if a full collection is performed, then the binary parameter \( \alpha_{i,r,p}^F = 1 \), and if a split or zero collection is performed, then the binary parameter \( \alpha_{i,r,p}^{SZ} = 1 \). This type of inequality was first proposed by Archetti et al. [8]. Let \( \gamma = (\gamma_1, \ldots, \gamma_n) \) be the values of the dual
variables associated with Constraints (3.39). The reduced cost $\bar{c}_{r,p}$ becomes

$$\bar{c}_{r,p} = c_{r,p} - \sum_{i \in r} (\delta_{i,p} \pi_i + \mu_i) - \sum_{i \in V_S} (2\alpha_{i,r,p}^F + \alpha_{i,r,p}^{SZ}) \gamma_i$$

(3.40)

To deal with the SMV inequalities, we need to modify the label-setting algorithm. First, define a new label $E_i$ that contain additional components as follows:

$$E_i = (\tau_i, N_i, V_i, G(r, q) = (F_i, SL_i, I_i), (\chi^j_i)_{j \in V_S}, Q_i)$$

where $\chi^j_i$ for all $j \in V_S$ are initialized to zero, $\chi^j_i = 1$ indicates that customer $j$ is forced to be full collection and $Q_i$ is the remaining capacity. Next, if $\gamma_j > 0$, we need to create two types of labels along edge $(i, j)$: type 1 label is for a zero or split collected is performed and type 2 label is for a full collection. The forward extension functions involving the new dual variables $\gamma_i$ and the new label components are as follows. For type 1 label, we have:

$$F_j = F_i + bc_{i,j} - \mu_j - \gamma_i;$$

$$\chi^k_j = \chi^k_i;$$

$$Q_j = Q_i.$$
and for type 2 label, we have:

\[ F_j = F_i + bc_{i,j} - \mu_j - 2\gamma_i; \]

\[ \chi^k_j = \begin{cases} 
1, & \text{if } k = j; \\
\chi^k_j, & \text{otherwise}; 
\end{cases} \]

\[ Q_j = Q_i - d_j. \]

Given a label \( E_i \), the minimum reduced cost is calculated as follows. The fixed cost \( f_r \) is the sum of \( F_i \) and \(-\sum_{k \in V_S: \chi^k_i = 1} d_k g_k\). With the values of \( g_k \in SL_i - \{v : \chi^v_i = 1\} \) and \( q \) equal to the remaining capacity \( Q_i \), we can perform the modified Algorithm 1 to achieve the value of \(-\sum_{k \in N_i - \{v : \chi^v_i = 1\}} \delta_k g_k\). The graph \( G(r, q) \) is illustrated in Figure 3.10, where we assume \( G(r, q) \) with \( 0 \leq q \leq \sum_{k \in V_S: \chi^k_i = 1} d_k \) equals a sufficiently large positive constant \( M \).

![Figure 3.10](image)

Figure 3.10: The graphic representation of \( G(r, q) \) after the introduction of SMV inequalities.

### 3.5.2 Search strategy

The branch-and-bound tree is explored according to a best-first policy; specifically, the “best” unexamined tree node is the one with the smallest lower bound, and would be given the highest priority. We have tested the depth-first policy in some
preliminary experiments and obtained inferior results in terms of the number of the optimally solved instances within the same amount of computation time.

3.5.3 Branching strategies

At each branch-and-bound node, we achieve an optimal solution of the LMP using column generation procedure and separation heuristics; this solution value is a lower bound at that node. If this lower bound is not less than the current upper bound, the associated node is pruned; otherwise, branching must take place. If the optimal solution of the LMP is integral and the optimal solution value is less than the current upper bound, we update the upper bound.

As explained in Desaulniers et al. [40], we can hardly branch on master problem variables \( \theta_{r,p} \) since fixing such variables at 0 requires preventing label-setting algorithms from generating the corresponding routes, significantly increasing the complexity of solving the pricing subproblem. Therefore, it is better to choose branching strategies compatible with the algorithms for the pricing subproblems, i.e., the pricing subproblems at the nodes resulting from such branchings could be solved in a way similar to the one used at their parent nodes. This requires that branching constraints do not change the structure of the pricing subproblem.

In our branch-and-price-and-cut algorithms, we choose four types of branching strategies that have been implemented in Desaulniers [39], namely branching on the total number of vehicles used, on the number of vehicles visiting each customer, on the total flow on each edge and on including or not including two consecutive edges in the vehicle routes.
3.6 Computational Experiments

3.6.1 Instances

To evaluate the branch-and-price-and-cut algorithm proposed in this chapter, we conducted experiments using the data set derived from the 56 benchmark VRPTW instances of Solomon [120], which are divided into six groups, namely R1, C1, RC1, R2, C2 and RC2. Each Solomon instance contains a designated depot and 100 customers, for a total of 101 vertices. From these 100-customer instances, we derived 25-customer and 50 customer instances by only considering the first 25 and 50 instances, respectively. For each of these instances, we consider three type of vehicle capacities, namely $Q = 30, 50$ and 100. Thus, we have 504 instances in total, which consists of 54 groups. Each group is identified by three parts separated by dashes (‘-’), i.e., the name of Solomon group, the number of customers ($n$) and the vehicle capacity ($Q$). For example, instance group R1-100-30 contains all instances with $n = 100$ and $Q = 30$ generated from Solomon instance group R1, and R101-100-30 is the identifier of the first instance in this group. As did by Desaulniers [39] and Archetti et al. [8], the Euclidean distance between any pair of vertices was rounded off to one decimal place. We set $a = 1$ and $b = Q/4$ for the weight-related cost function $f(w) = a \times w + b$. This implies that one dollar is charged per unit distance per unit weight and the vehicle weight equals one-quarter of the weight that a vehicle can carry. Note that if we set $a = 0$ and $b = 1$, the instances becomes SDVRPTW instances used in Gendreau et al. [57], Desaulniers [39], Archetti et al. [8]. All instances as well as detailed experimental results are available in the online supplement at:

www.computational-logistics.org/orlib/scvrptwl
3.6.2 Experimental Setup

Our algorithm was coded in Java and all experiments were conducted on a Dell server with an Intel Xeon E5520 2.26 GHz CPU, 8 GB RAM and Linux operating system. The linear programming models were solved by simplex algorithm implemented by ILOG CPLEX 12.0. Computation times reported are in CPU seconds on this server.

We imposed a time limit of 3,600 seconds on each execution of the branch-and-price-and-cut algorithm. However, when the time limit is reached, we do not terminate the algorithm until it finishes processing the current branch-and-bound node. The parameters used in our heuristic column generator were fixed as: \( \text{maxCol} = 1000, \text{maxIter} = 25 \times n \) and \( \eta = 0.15 \).

3.6.3 Results on the SDVRPTW instances

First, we applied our branch-and-price-and-cut algorithm to solve the SDVRPTW instances. At the root node of the branch-and-bound tree, we solved the linear relaxation of the problem without introducing any valid inequality using the column generation procedure. The linear relaxation results are mainly determined by the performance of the label-setting algorithm. We compare our results with the results taken from Desaulniers \cite{39} and Archetti et al. \cite{8} in Table 3.1.

The results produced by our algorithm are presented in the blocks “New”; the column “# inst” gives the number of instances in the Solomon instance group; the columns “# solved” give the numbers of optimally solved instances within the time limit; and the columns “Time” show the average computation times. Since the experimental environments (Language C/C++, a Linux PC equipped with a Pentium D processor clocked at 2.8 GHz and CPLEX 10.1.1) used by Desaulniers \cite{39}, Archetti et al. \cite{8} are quite different from ours, we can not directly judge whether our results are better than theirs. We can only say that we have achieved
the optimal solutions to the linear relaxations of all SDVRPTW instances used by Desaulniers [39], Archetti et al. [8].
<table>
<thead>
<tr>
<th>η</th>
<th>group</th>
<th># inst</th>
<th># solved</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>Q-50</td>
<td>New</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>119</td>
<td></td>
<td>Downham (2001)</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>126</td>
<td></td>
<td>Archetti et al. (2011)</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>133</td>
<td></td>
<td>New</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>140</td>
<td></td>
<td>Downham (2001)</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>148</td>
<td></td>
<td>Archetti et al. (2011)</td>
<td>12</td>
<td>&lt;1</td>
</tr>
</tbody>
</table>

Table 3.1: Linear Relaxation Results on the SDVRPTW Instances.
From Table 3.1, we find that the algorithm proposed in Archetti et al. [8] failed to optimally solve the linear relaxation of four instances in group R2-100-100. However, this article does not reveal the names of these four instances. So we present in Table 3.2 the optimal linear relaxation values (LP) and associated computation times (LP time) of all instances in group R2-100-100.

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>Q</th>
<th>LP</th>
<th>LP time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R201</td>
<td>100</td>
<td>100</td>
<td>1300.6</td>
<td>34.5</td>
</tr>
<tr>
<td>R202</td>
<td>100</td>
<td>100</td>
<td>1231.2</td>
<td>59.0</td>
</tr>
<tr>
<td>R203</td>
<td>100</td>
<td>100</td>
<td>1158.8</td>
<td>121.0</td>
</tr>
<tr>
<td>R204</td>
<td>100</td>
<td>100</td>
<td>1126.3</td>
<td>496.1</td>
</tr>
<tr>
<td>R205</td>
<td>100</td>
<td>100</td>
<td>1205.6</td>
<td>64.0</td>
</tr>
<tr>
<td>R206</td>
<td>100</td>
<td>100</td>
<td>1169.3</td>
<td>83.9</td>
</tr>
<tr>
<td>R207</td>
<td>100</td>
<td>100</td>
<td>1144.0</td>
<td>251.3</td>
</tr>
<tr>
<td>R208</td>
<td>100</td>
<td>100</td>
<td>1122.0</td>
<td>643.1</td>
</tr>
<tr>
<td>R209</td>
<td>100</td>
<td>100</td>
<td>1158.8</td>
<td>105.1</td>
</tr>
<tr>
<td>R210</td>
<td>100</td>
<td>100</td>
<td>1166.3</td>
<td>105.9</td>
</tr>
<tr>
<td>R211</td>
<td>100</td>
<td>100</td>
<td>1120.9</td>
<td>279.7</td>
</tr>
</tbody>
</table>

Table 3.2: Linear Relaxation Results on the SDVRPTW Instances in Group R2-100-100.

During the experiments, we found that some instances violate the triangle inequality due to rounding the distance to one decimal place. Consequently, the optimal solutions reported in Desaulniers [39], Archetti et al. [8] for some SD-VRPTW instances are not truly optimal. To resolve this issue, we applied a shortest path algorithm to update the distance matrix and make it satisfy the triangle inequality. Then, we solved all SDVRPTW instances again using our algorithm. The optimal solution values of the 262 SDVRPTW instances obtained by previous articles can be found at:

http://www.gerad.ca/~guyd/sdvrptw.html

Based on our computational results, we divided the instances into five categories:

Category 1: The optimal solution values are smaller than those reported in Archetti et al. [8];
Category 2: The optimal solutions are not reported in Archetti et al. \[8\] but have been found by our algorithm;

Category 3: The optimal solutions were reported in Archetti et al. \[8\] but have not been found by our algorithm;

Category 4: The optimal solution values are the same as those reported in Archetti et al. \[8]\;

Category 5: The optimal solutions have not been found by any algorithm.

Our algorithm achieved optimal solutions for 264 out of 504 SDVRPTW instances. Table 3.3 presents the detailed integer results for the 52 instances belonging to Categories 1 and 2, including the number of vehicles used (# vehicles), the number of split customers (# splits), the number of branch-and-bound nodes (# nodes), the number of added cuts (# cuts), the integer solution value (IP) and the consumed computation time (Time). The instances contained in Category 3 are R102-50-50, C103-50-100, C202-50-100, C205-50-100, C102-100-100, C205-100-100 and C206-100-100.

### 3.6.4 Results on the SCVRPTWL instances

Next, we tried to solve all SCVRPTWL instances to optimality using our branch-and-price-and-cut algorithm. These instances use the updated distance matrix that satisfies the triangle inequality. At the beginning of the algorithm, we solved the linear relaxation of the problem that contains all \( n \) SMV inequalities and does not consider any \( k \)-path inequality. The linear relaxation results are reported in Table 3.4 which show that a lower bound for each instance was achieved within the time limit.

Table 3.5 presents a summary of the integer solution results of the SCVRPTWL instances. All columns except the first three columns give the average values over
<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>Q</th>
<th># vehicles</th>
<th># splits</th>
<th># nodes</th>
<th># cuts</th>
<th>IP</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C201</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>1</td>
<td>69</td>
<td>909.8</td>
<td>2.1</td>
</tr>
<tr>
<td>C202</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>1</td>
<td>136</td>
<td>909.8</td>
<td>2.8</td>
</tr>
<tr>
<td>C203</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>10</td>
<td>114</td>
<td>909.8</td>
<td>8.6</td>
</tr>
<tr>
<td>C204</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>8</td>
<td>63</td>
<td>909.8</td>
<td>10.2</td>
</tr>
<tr>
<td>C205</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>9</td>
<td>102</td>
<td>909.8</td>
<td>16.2</td>
</tr>
<tr>
<td>C206</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>6</td>
<td>1</td>
<td>52</td>
<td>909.8</td>
<td>2.1</td>
</tr>
<tr>
<td>C207</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>2</td>
<td>55</td>
<td>909.8</td>
<td>3.2</td>
</tr>
<tr>
<td>C208</td>
<td>25</td>
<td>30</td>
<td>16</td>
<td>3</td>
<td>9</td>
<td>40</td>
<td>630.2</td>
<td>4.8</td>
</tr>
<tr>
<td>C209</td>
<td>25</td>
<td>30</td>
<td>10</td>
<td>3</td>
<td>23</td>
<td>59</td>
<td>600.0</td>
<td>8.1</td>
</tr>
<tr>
<td>C210</td>
<td>25</td>
<td>30</td>
<td>10</td>
<td>2</td>
<td>27</td>
<td>49</td>
<td>600.0</td>
<td>9.4</td>
</tr>
<tr>
<td>C211</td>
<td>25</td>
<td>30</td>
<td>10</td>
<td>3</td>
<td>25</td>
<td>42</td>
<td>600.0</td>
<td>10.2</td>
</tr>
<tr>
<td>RC201</td>
<td>25</td>
<td>50</td>
<td>11</td>
<td>1</td>
<td>33</td>
<td>130</td>
<td>946.0</td>
<td>17.1</td>
</tr>
<tr>
<td>C101</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>298.1</td>
<td>1.6</td>
</tr>
<tr>
<td>C102</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>17</td>
<td>298.1</td>
<td>3.9</td>
</tr>
<tr>
<td>C103</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>298.1</td>
<td>3.4</td>
</tr>
<tr>
<td>C104</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>13</td>
<td>298.1</td>
<td>3.8</td>
</tr>
<tr>
<td>C105</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>9</td>
<td>20</td>
<td>298.1</td>
<td>4.4</td>
</tr>
<tr>
<td>C106</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>363.5</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>C107</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>72</td>
<td>359.9</td>
<td>1.4</td>
</tr>
<tr>
<td>C108</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>1</td>
<td>17</td>
<td>37</td>
<td>358.7</td>
<td>13.8</td>
</tr>
<tr>
<td>C109</td>
<td>25</td>
<td>100</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>72</td>
<td>358.7</td>
<td>11.4</td>
</tr>
<tr>
<td>RC202</td>
<td>25</td>
<td>100</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>534.0</td>
<td>1.3</td>
</tr>
<tr>
<td>RC203</td>
<td>25</td>
<td>100</td>
<td>6</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>526.2</td>
<td>2.4</td>
</tr>
<tr>
<td>R101</td>
<td>50</td>
<td>50</td>
<td>15</td>
<td>3</td>
<td>61</td>
<td>63</td>
<td>1,190.7</td>
<td>10.9</td>
</tr>
<tr>
<td>C108</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>6</td>
<td>1</td>
<td>248</td>
<td>1,011.8</td>
<td>18.2</td>
</tr>
<tr>
<td>C201</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>8</td>
<td>15</td>
<td>290</td>
<td>1,159.4</td>
<td>99.4</td>
</tr>
<tr>
<td>C202</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>8</td>
<td>1</td>
<td>260</td>
<td>1,156.9</td>
<td>19.9</td>
</tr>
<tr>
<td>C203</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>8</td>
<td>15</td>
<td>273</td>
<td>1,156.9</td>
<td>268.2</td>
</tr>
<tr>
<td>C204</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>9</td>
<td>1</td>
<td>592</td>
<td>1,156.9</td>
<td>55.8</td>
</tr>
<tr>
<td>C205</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>8</td>
<td>5</td>
<td>558</td>
<td>1,156.9</td>
<td>311.1</td>
</tr>
<tr>
<td>C206</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>8</td>
<td>19</td>
<td>232</td>
<td>1,156.9</td>
<td>199.8</td>
</tr>
<tr>
<td>C207</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>9</td>
<td>5</td>
<td>1,076</td>
<td>1,156.9</td>
<td>492.2</td>
</tr>
<tr>
<td>R101</td>
<td>50</td>
<td>100</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1,043.8</td>
<td>1.1</td>
</tr>
<tr>
<td>R102</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>0</td>
<td>316</td>
<td>39</td>
<td>918.1</td>
<td>173.2</td>
</tr>
<tr>
<td>R103</td>
<td>50</td>
<td>100</td>
<td>8</td>
<td>1</td>
<td>1,267</td>
<td>176</td>
<td>804.1</td>
<td>2,404</td>
</tr>
<tr>
<td>C101</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>0</td>
<td>111</td>
<td>451</td>
<td>587.5</td>
<td>1,487.4</td>
</tr>
<tr>
<td>C102</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>1</td>
<td>37</td>
<td>425</td>
<td>584.5</td>
<td>1,061.9</td>
</tr>
<tr>
<td>C103</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>2</td>
<td>115</td>
<td>385</td>
<td>587.5</td>
<td>1,904.9</td>
</tr>
<tr>
<td>C104</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>3</td>
<td>181</td>
<td>412</td>
<td>587.5</td>
<td>2,175.3</td>
</tr>
<tr>
<td>C105</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>3</td>
<td>281</td>
<td>139</td>
<td>587.5</td>
<td>1,718.7</td>
</tr>
<tr>
<td>C106</td>
<td>50</td>
<td>100</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>154</td>
<td>584.0</td>
<td>80.9</td>
</tr>
<tr>
<td>R205</td>
<td>50</td>
<td>100</td>
<td>8</td>
<td>1</td>
<td>49</td>
<td>120</td>
<td>758.8</td>
<td>418.1</td>
</tr>
<tr>
<td>R101</td>
<td>50</td>
<td>100</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>1,038.4</td>
<td>89.9</td>
</tr>
<tr>
<td>C101</td>
<td>50</td>
<td>50</td>
<td>29</td>
<td>10</td>
<td>1</td>
<td>860</td>
<td>1,599.5</td>
<td>10.0</td>
</tr>
<tr>
<td>C102</td>
<td>50</td>
<td>50</td>
<td>29</td>
<td>10</td>
<td>1</td>
<td>2,689</td>
<td>1,599.5</td>
<td>260.3</td>
</tr>
<tr>
<td>C103</td>
<td>50</td>
<td>50</td>
<td>29</td>
<td>7</td>
<td>1</td>
<td>1,214</td>
<td>1,599.5</td>
<td>75.7</td>
</tr>
<tr>
<td>C104</td>
<td>50</td>
<td>50</td>
<td>29</td>
<td>7</td>
<td>1</td>
<td>683</td>
<td>1,599.5</td>
<td>21.1</td>
</tr>
<tr>
<td>C105</td>
<td>50</td>
<td>50</td>
<td>29</td>
<td>9</td>
<td>1</td>
<td>767</td>
<td>1,599.5</td>
<td>35.5</td>
</tr>
<tr>
<td>C106</td>
<td>50</td>
<td>50</td>
<td>29</td>
<td>10</td>
<td>2</td>
<td>1,547</td>
<td>1,598.4</td>
<td>137.7</td>
</tr>
<tr>
<td>C204</td>
<td>50</td>
<td>50</td>
<td>18</td>
<td>9</td>
<td>5</td>
<td>277</td>
<td>1,156.9</td>
<td>213.3</td>
</tr>
<tr>
<td>R201</td>
<td>50</td>
<td>100</td>
<td>8</td>
<td>0</td>
<td>1,557</td>
<td>231</td>
<td>843.0</td>
<td>2,709.8</td>
</tr>
</tbody>
</table>

Table 3.3: Integer Solution Results for the SDVRPTW Instances in Categories 1 and 2.
Table 3.4: Linear Relaxation Results on the SCVRPTWL Instances.

<table>
<thead>
<tr>
<th>n</th>
<th>Solomon group</th>
<th># inst</th>
<th># solved</th>
<th>Time</th>
<th># solved</th>
<th>Time</th>
<th># solved</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>R1</td>
<td>12</td>
<td>12</td>
<td>0.6</td>
<td>12</td>
<td>0.8</td>
<td>12</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>9</td>
<td>9</td>
<td>0.7</td>
<td>9</td>
<td>0.9</td>
<td>9</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>RC1</td>
<td>8</td>
<td>8</td>
<td>0.6</td>
<td>8</td>
<td>0.7</td>
<td>8</td>
<td>1.5</td>
</tr>
<tr>
<td>50</td>
<td>R1</td>
<td>12</td>
<td>12</td>
<td>2.0</td>
<td>12</td>
<td>3.0</td>
<td>12</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>9</td>
<td>9</td>
<td>1.7</td>
<td>9</td>
<td>2.2</td>
<td>9</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>RC1</td>
<td>8</td>
<td>8</td>
<td>1.3</td>
<td>8</td>
<td>2.0</td>
<td>8</td>
<td>4.0</td>
</tr>
<tr>
<td>100</td>
<td>R1</td>
<td>12</td>
<td>12</td>
<td>14.9</td>
<td>12</td>
<td>23.3</td>
<td>12</td>
<td>65.0</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>9</td>
<td>9</td>
<td>6.2</td>
<td>9</td>
<td>8.6</td>
<td>9</td>
<td>21.7</td>
</tr>
<tr>
<td></td>
<td>RC1</td>
<td>8</td>
<td>8</td>
<td>9.1</td>
<td>8</td>
<td>13.8</td>
<td>8</td>
<td>59.2</td>
</tr>
<tr>
<td>25</td>
<td>R2</td>
<td>11</td>
<td>11</td>
<td>0.8</td>
<td>11</td>
<td>1.1</td>
<td>11</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>8</td>
<td>8</td>
<td>0.7</td>
<td>8</td>
<td>0.8</td>
<td>8</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>RC2</td>
<td>8</td>
<td>8</td>
<td>0.6</td>
<td>8</td>
<td>0.9</td>
<td>8</td>
<td>1.8</td>
</tr>
<tr>
<td>50</td>
<td>R2</td>
<td>11</td>
<td>11</td>
<td>2.6</td>
<td>11</td>
<td>4.7</td>
<td>11</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>8</td>
<td>8</td>
<td>1.5</td>
<td>8</td>
<td>2.1</td>
<td>8</td>
<td>5.7</td>
</tr>
<tr>
<td></td>
<td>RC2</td>
<td>8</td>
<td>8</td>
<td>1.5</td>
<td>8</td>
<td>2.4</td>
<td>8</td>
<td>5.4</td>
</tr>
<tr>
<td>100</td>
<td>R2</td>
<td>11</td>
<td>11</td>
<td>21.3</td>
<td>11</td>
<td>41.1</td>
<td>11</td>
<td>168.9</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>8</td>
<td>8</td>
<td>6.2</td>
<td>8</td>
<td>10.3</td>
<td>8</td>
<td>35.7</td>
</tr>
<tr>
<td></td>
<td>RC2</td>
<td>8</td>
<td>8</td>
<td>12.8</td>
<td>8</td>
<td>190.4</td>
<td>8</td>
<td>476.7</td>
</tr>
</tbody>
</table>

the optimally solved instances. We denote by LP and LPC the optimal values of
the linear relaxations (at the root node) with and without the k-path inequalities,
respectively. The value of “LP gap (%)” (respectively, “LPC gap (%)”) for each
solved instance was calculated by (IP − LP)/IP (respectively, (IP − LPC)/IP),
where IP represents the optimal integer solution value. The average times used
to produce LP, LPC and IP are reported in the columns “LP time”, “LPC time”
and “IP time”, respectively.
<table>
<thead>
<tr>
<th>Instance group</th>
<th># inst</th>
<th># solved</th>
<th># vehicles</th>
<th># splits</th>
<th>LP gap (%)</th>
<th>LP time</th>
<th>LPC gap (%)</th>
<th>LPC time</th>
<th>IP time</th>
<th># nodes</th>
<th># cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1-25-30</td>
<td>12</td>
<td>12</td>
<td>13.4</td>
<td>1.8</td>
<td>0.39</td>
<td>0.6</td>
<td>0.13</td>
<td>1.4</td>
<td>117.2</td>
<td>336.5</td>
<td>66.2</td>
</tr>
<tr>
<td>R1-25-50</td>
<td>12</td>
<td>12</td>
<td>9.4</td>
<td>0.9</td>
<td>0.33</td>
<td>0.8</td>
<td>0.20</td>
<td>1.0</td>
<td>6.6</td>
<td>19.8</td>
<td>43</td>
</tr>
<tr>
<td>R1-25-100</td>
<td>12</td>
<td>12</td>
<td>7.3</td>
<td>0.4</td>
<td>0.10</td>
<td>0.8</td>
<td>0.10</td>
<td>0.8</td>
<td>1.1</td>
<td>1.7</td>
<td>0.0</td>
</tr>
<tr>
<td>R1-25-100</td>
<td>12</td>
<td>12</td>
<td>10.0</td>
<td>1.8</td>
<td>1.53</td>
<td>0.8</td>
<td>0.13</td>
<td>1.9</td>
<td>49.3</td>
<td>107.3</td>
<td>46.2</td>
</tr>
<tr>
<td>R1-25-100</td>
<td>12</td>
<td>12</td>
<td>9.4</td>
<td>0.9</td>
<td>0.33</td>
<td>0.8</td>
<td>0.20</td>
<td>1.0</td>
<td>6.6</td>
<td>19.8</td>
<td>43</td>
</tr>
<tr>
<td>R2-25-30</td>
<td>12</td>
<td>12</td>
<td>10.0</td>
<td>1.8</td>
<td>1.53</td>
<td>0.8</td>
<td>0.13</td>
<td>1.9</td>
<td>49.3</td>
<td>107.3</td>
<td>46.2</td>
</tr>
<tr>
<td>R2-25-50</td>
<td>12</td>
<td>12</td>
<td>9.4</td>
<td>0.9</td>
<td>0.33</td>
<td>0.8</td>
<td>0.20</td>
<td>1.0</td>
<td>6.6</td>
<td>19.8</td>
<td>43</td>
</tr>
<tr>
<td>R2-25-100</td>
<td>12</td>
<td>12</td>
<td>7.3</td>
<td>0.4</td>
<td>0.10</td>
<td>0.8</td>
<td>0.10</td>
<td>0.8</td>
<td>1.1</td>
<td>1.7</td>
<td>0.0</td>
</tr>
<tr>
<td>R2-50-100</td>
<td>12</td>
<td>12</td>
<td>10.0</td>
<td>1.8</td>
<td>1.53</td>
<td>0.8</td>
<td>0.13</td>
<td>1.9</td>
<td>49.3</td>
<td>107.3</td>
<td>46.2</td>
</tr>
<tr>
<td>R2-50-30</td>
<td>12</td>
<td>12</td>
<td>9.4</td>
<td>0.9</td>
<td>0.33</td>
<td>0.8</td>
<td>0.20</td>
<td>1.0</td>
<td>6.6</td>
<td>19.8</td>
<td>43</td>
</tr>
<tr>
<td>R2-50-50</td>
<td>12</td>
<td>12</td>
<td>7.3</td>
<td>0.4</td>
<td>0.10</td>
<td>0.8</td>
<td>0.10</td>
<td>0.8</td>
<td>1.1</td>
<td>1.7</td>
<td>0.0</td>
</tr>
<tr>
<td>R2-50-100</td>
<td>12</td>
<td>12</td>
<td>10.0</td>
<td>1.8</td>
<td>1.53</td>
<td>0.8</td>
<td>0.13</td>
<td>1.9</td>
<td>49.3</td>
<td>107.3</td>
<td>46.2</td>
</tr>
<tr>
<td>R2-100-100</td>
<td>12</td>
<td>12</td>
<td>9.4</td>
<td>0.9</td>
<td>0.33</td>
<td>0.8</td>
<td>0.20</td>
<td>1.0</td>
<td>6.6</td>
<td>19.8</td>
<td>43</td>
</tr>
<tr>
<td>R2-100-30</td>
<td>12</td>
<td>12</td>
<td>7.3</td>
<td>0.4</td>
<td>0.10</td>
<td>0.8</td>
<td>0.10</td>
<td>0.8</td>
<td>1.1</td>
<td>1.7</td>
<td>0.0</td>
</tr>
<tr>
<td>R2-100-50</td>
<td>12</td>
<td>12</td>
<td>10.0</td>
<td>1.8</td>
<td>1.53</td>
<td>0.8</td>
<td>0.13</td>
<td>1.9</td>
<td>49.3</td>
<td>107.3</td>
<td>46.2</td>
</tr>
<tr>
<td>R2-100-100</td>
<td>12</td>
<td>12</td>
<td>9.4</td>
<td>0.9</td>
<td>0.33</td>
<td>0.8</td>
<td>0.20</td>
<td>1.0</td>
<td>6.6</td>
<td>19.8</td>
<td>43</td>
</tr>
</tbody>
</table>

Table 3.5: Summary of the Integer Solution Results to the SCVRPTWL Instances.
Our algorithm optimally solved 188 out of 504 SCVRPTWL instances, where 130, 57, and 1 instances contain 25, 50, and 100 customers, respectively. Further, we summarized the information on the number of solved instances and the number of split customers in Tables 3.6 and 3.7. From Table 3.6, we can see that 48, 23, 29, 43, 18, and 27 solved instances were derived from Solomon groups R1, C1, RC1, R2, C2, and RC2, respectively. Moreover, this table also clearly shows that the instances with greater $Q$ are easier to be solved. Table 3.7 implies that with the increase of the vehicle capacity $Q$, the possibility of splitting customers becomes smaller. The detailed integer solution results for all optimally solved instances are given in Tables 3.8, 3.9, and 3.10.

<table>
<thead>
<tr>
<th>Solomon group</th>
<th>n</th>
<th>$Q = 30$</th>
<th>$Q = 50$</th>
<th>$Q = 100$</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>25</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
<td>2</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>25</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RC1</td>
<td>25</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R2</td>
<td>25</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>25</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RC2</td>
<td>25</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td>24</td>
<td>68</td>
<td>96</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.6: Summary on the Number of Solved Instances.

### 3.7 Conclusions

This chapter introduces a new extension of the SDVRPTW in which the travel cost per unit distance is charged based on a linear function of the vehicle weight;
Table 3.7: Summary on the Number of Split Customers.

<table>
<thead>
<tr>
<th>Solomon group</th>
<th>n</th>
<th>Q = 30</th>
<th>Q = 50</th>
<th>Q = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>25</td>
<td>–</td>
<td>1.8</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>–</td>
<td>2.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C1</td>
<td>25</td>
<td>–</td>
<td>1.1</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>–</td>
<td>–</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RC1</td>
<td>25</td>
<td>–</td>
<td>1.1</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>–</td>
<td>6.5</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>R2</td>
<td>25</td>
<td>–</td>
<td>2.2</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>–</td>
<td>–</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>5.0</td>
</tr>
<tr>
<td>C2</td>
<td>25</td>
<td>–</td>
<td>6.0</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>–</td>
<td>–</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RC2</td>
<td>25</td>
<td>–</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>–</td>
<td>6.1</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

this extension is called the split-collection vehicle routing problem with time windows and linear weight-related cost (SCVRPTWL). We devised an exact branch-and-price-and-cut algorithm to solve the problem, where the pricing subproblem is a resource-constrained elementary least-cost path problem. The effectiveness of the branch-and-price-and-cut algorithm heavily relies on the method for solving the pricing subproblem. We observed that at least one of the optimal solutions to the pricing subproblem must correspond to an extreme collection pattern; this help us reduce the feasible region significantly. To solve this new type of pricing subproblem, we designed a tailored and novel label-setting algorithm that integrates specific labels and dominance rules. We applied our branch-and-price-and-cut algorithm to solve the instances of both the SDVRPTW and SCVRPTWL.

The reported computational results reveal that our algorithm achieved optimal solutions for 264 SDVRPTW instances and 188 SCVRPTWL instances within one hour of computation time. The existing best exact algorithm, namely the enhanced branch-and-price-and-cut algorithm proposed by Archetti et al. [8], only produced optimal solutions for 262 SDVRPTW instances. Since the SCVRPTWL is a new problem and has not been tackled by any exiting algorithm, the experiments and analysis presented in this study serves as benchmarks for
<table>
<thead>
<tr>
<th>Instance group</th>
<th>Instance</th>
<th>n vehicles</th>
<th>n splits</th>
<th>LP</th>
<th>LP time</th>
<th>LPs</th>
<th>LS+ time</th>
<th>IP</th>
<th>IP time</th>
<th># nodes</th>
<th># cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1-25-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R2-30-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R3-25-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R4-25-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>C1-25-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R1-30-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R2-35-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R3-30-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R4-30-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>C1-30-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R1-35-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R2-40-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
<tr>
<td>R3-40-30</td>
<td>14</td>
<td>2</td>
<td>14,530.6</td>
<td>0.6</td>
<td>10,762.8</td>
<td>1.0</td>
<td>17,374.9</td>
<td>0.7</td>
<td>17,423.2</td>
<td>2.9</td>
<td>639</td>
</tr>
</tbody>
</table>

Table 3.8: Optimal Integer Solutions for the SCVRPTWL instances (Part I).
<table>
<thead>
<tr>
<th>Instance group</th>
<th>Instance</th>
<th>α</th>
<th>vehicles</th>
<th>α-split</th>
<th>IP</th>
<th>IP time</th>
<th>LP</th>
<th>LP time</th>
<th>IP</th>
<th>IP time</th>
<th># nodes</th>
<th># cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2-25-50</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>20,914.0</td>
<td>0.8</td>
<td>20,914.0</td>
<td>0.8</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>20,732.4</td>
<td>1.2</td>
<td>20,732.4</td>
<td>1.2</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>20,434.3</td>
<td>1.3</td>
<td>20,434.3</td>
<td>1.3</td>
<td>1</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>1</td>
<td>20,434.3</td>
<td>1.3</td>
<td>20,434.3</td>
<td>1.3</td>
<td>1</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>20,563.0</td>
<td>0.8</td>
<td>20,563.0</td>
<td>0.8</td>
<td>1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>20,563.0</td>
<td>1.4</td>
<td>20,563.0</td>
<td>1.4</td>
<td>1</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>20,563.0</td>
<td>1.4</td>
<td>20,563.0</td>
<td>1.4</td>
<td>1</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>10</td>
<td>0</td>
<td>20,434.3</td>
<td>1.5</td>
<td>20,434.3</td>
<td>1.5</td>
<td>1</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC2-25-50</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>32,840.0</td>
<td>1.6</td>
<td>32,840.0</td>
<td>1.6</td>
<td>1</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>32,840.0</td>
<td>1.6</td>
<td>32,840.0</td>
<td>1.6</td>
<td>1</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>12</td>
<td>2</td>
<td>32,840.0</td>
<td>2.3</td>
<td>32,405.0</td>
<td>1.1</td>
<td>1</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>12</td>
<td>1</td>
<td>32,840.0</td>
<td>2.3</td>
<td>32,405.0</td>
<td>1.1</td>
<td>1</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>12</td>
<td>1</td>
<td>32,840.0</td>
<td>2.3</td>
<td>32,405.0</td>
<td>1.1</td>
<td>1</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
<td>1</td>
<td>32,840.0</td>
<td>2.3</td>
<td>32,405.0</td>
<td>1.1</td>
<td>1</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1-25-100</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>7</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>27,102.8</td>
<td>0.4</td>
<td>27,102.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1-25-100</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>21,034.0</td>
<td>1.1</td>
<td>21,034.0</td>
<td>1.1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2-25-100</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>35,908.0</td>
<td>1.5</td>
<td>35,908.0</td>
<td>1.5</td>
<td>1</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3-25-100</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>23,478.1</td>
<td>1.7</td>
<td>23,478.1</td>
<td>1.7</td>
<td>1</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.9: Optimal Integer Solutions for the SCVRPTWL instances (Part II).
<table>
<thead>
<tr>
<th>Instance group</th>
<th>Instance</th>
<th>m</th>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC2-25-100</td>
<td>RC2-25-100</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>RC1-50-50</td>
<td>RC1-50-50</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>RC2-50-50</td>
<td>RC2-50-50</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>12</td>
<td>10</td>
<td>1</td>
<td>12</td>
<td>10</td>
<td>1</td>
<td>12</td>
<td>10</td>
<td>1</td>
<td>12</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>R1-50-50</td>
<td>R1-50-50</td>
<td>1</td>
<td>14</td>
<td>1</td>
<td>16</td>
<td>14</td>
<td>1</td>
<td>16</td>
<td>14</td>
<td>1</td>
<td>16</td>
<td>14</td>
<td>1</td>
<td>16</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>C1-50-100</td>
<td>C1-50-100</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>20</td>
<td>18</td>
<td>1</td>
<td>20</td>
<td>18</td>
<td>1</td>
<td>20</td>
<td>18</td>
<td>1</td>
<td>20</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>RC1-50-100</td>
<td>RC1-50-100</td>
<td>1</td>
<td>22</td>
<td>1</td>
<td>24</td>
<td>22</td>
<td>1</td>
<td>24</td>
<td>22</td>
<td>1</td>
<td>24</td>
<td>22</td>
<td>1</td>
<td>24</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>R2-50-100</td>
<td>R2-50-100</td>
<td>1</td>
<td>26</td>
<td>1</td>
<td>28</td>
<td>26</td>
<td>1</td>
<td>28</td>
<td>26</td>
<td>1</td>
<td>28</td>
<td>26</td>
<td>1</td>
<td>28</td>
<td>26</td>
<td>1</td>
</tr>
<tr>
<td>C2-50-100</td>
<td>C2-50-100</td>
<td>1</td>
<td>30</td>
<td>1</td>
<td>32</td>
<td>30</td>
<td>1</td>
<td>32</td>
<td>30</td>
<td>1</td>
<td>32</td>
<td>30</td>
<td>1</td>
<td>32</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>RC2-100-100</td>
<td>RC2-100-100</td>
<td>1</td>
<td>34</td>
<td>1</td>
<td>36</td>
<td>34</td>
<td>1</td>
<td>36</td>
<td>34</td>
<td>1</td>
<td>36</td>
<td>34</td>
<td>1</td>
<td>36</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>R2-100-100</td>
<td>R2-100-100</td>
<td>1</td>
<td>38</td>
<td>1</td>
<td>40</td>
<td>38</td>
<td>1</td>
<td>40</td>
<td>38</td>
<td>1</td>
<td>40</td>
<td>38</td>
<td>1</td>
<td>40</td>
<td>38</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.10: Optimal Integer Solutions for the SCVRPTWL instances (Part III).
future researchers.

Since our branch-and-price-and-cut algorithm only optimally solved around one-third of the total benchmark SCVRPTWL instances, there is much space to improve the solution procedure. Furthermore, we may investigate other vehicle routing models that incorporates the linear weight-related cost or other types of cost functions, e.g., piecewise linear function of the vehicle weight.
Chapter 4

Vehicle Routing Problem with
Stochastic Demands and
Toll-by-Weight Scheme

4.1 Introduction

By the end of 2013, over twenty eight Chinese provinces have implemented the
toll-by-weight scheme in which expressway tolls are levied according to vehicle
weight and traveling distance. The toll-by-weight scheme usually consists of two
parts, namely, regular toll scheme and penalty scheme. In China, vehicles are
classified into different types according to the number of axles in their chassis.
Each vehicle type has a prescribed and province-specific weight limit $Q_L$. The
weight portion less than or equal to $Q_L$ is charged based on the regular toll
scheme while the excess weight is charged according to the penalty scheme. Each
toll-by-weight scheme can be represented by a piecewise function $c(w)$, called toll
function, which equals the toll paid by the vehicle with weight $w$ for one unit
distance. Function (4.1) shows the toll-by-weight scheme used in Fujian province
for the vehicle with $Q_L = 40$, where each piece is either linear or quadratic. The
first two pieces in this function correspond to the regular toll scheme and the last
one is the penalty scheme. When a vehicle has a weight greater than \(1.3Q_L\), it is forbidden to enter the expressways.

\[
c(w) = \begin{cases} 
0.09w, & \text{if } w \leq 10 \\
-0.0015w^2 + 0.12w - 0.15, & \text{if } 10 \leq w \leq Q_L \\
0.09w - 0.0015Q_L^2 + 0.03Q_L - 0.15, & \text{if } Q_L \leq w \leq 1.3Q_L
\end{cases}
\]

The toll-by-weight scheme has tremendous influence on Chinese transportation industry. Before the implementation of this new toll scheme, tolls in Chinese expressways have been levied based on the tonnage or seating capacity of the vehicle and traveling distance. Under this toll scheme, a vehicle is charged the same toll regardless of its weight, which violates the principle of equity where more loads should incur greater cost. In addition, this toll scheme motivates transportation companies to overload their vehicles to the greatest extent for economic benefits. The overloading brings about several obvious and serious issues, such as the damage of the expressways, the increasing safety risk to drivers and other expressway users, and the disruption of the transportation market. Now under the toll-by-weight scheme, transportation companies have to load their vehicles at a reasonable level and pay the tolls in accordance with the freight weight they transport.

Compared to the classical vehicle routing problem (VRP) \([125]\), the VRP with toll-by-weight scheme (VRPTBW) has received relatively little attention in literature despite its numerous practical applications in Chinese transportation system. The VRPTBW was first mentioned by Shen et al. \([119]\), who incorporated the toll-by-weight scheme into the classical VRP and implemented a simulated annealing algorithm to solve the problem. Subsequently, Zhang et al. \([131]\) introduced a VRP variant that considers the toll-by-weight scheme and a single vehicle. They designed a branch-and-bound algorithm to exactly solve their
problem. Most recently, Luo et al. [90] devised a branch-and-price-and-cut algorithm to solve a multiple traveling repairmen problem with distance constraints (MTRPD); this problem is essentially a VRP variant with linear toll function. The workover rig routing problem in Ribeiro et al. [107, 108] can be viewed as a variant of the MTRPD although it arises in the operations of onshore oil fields.

In many real-life situations [45, 21, 103], customers have stochastic demands, which necessitates the investigation of the VRP with stochastic demands (VRPSD). The stochastic demand may lead to a route failure when the demand realization exceeds the remaining capacity of the vehicle. Upon a route failure, the vehicle takes a recourse action, i.e., returns to the depot to unload all collected cargoes and then continues to visit the remaining customers. Unlike the classical VRP, the VRPSD aims to minimize the total expected traveling distance of all involved vehicles. In recent years, an increasing number of researchers have devoted to studying the VRPSD; for more details of this problem, see Dror et al. [45], Bertsimas [21], Dror [46].

This chapter addresses a new VRP variant yielded by combining the VRPSD and VRPTBW, which is therefore termed the VRP with stochastic demands and toll-by-weight scheme (VRPSD-TBW). As stated in Psaraftis [103], three main solution concepts for the VRPSD are available in literature, which are chance-constrained programming (CCP), reoptimization and a priori optimization. The chance-constrained programming model tries to find a set of routes that minimizes the total expected cost while guaranteeing that the probability of route failure is not greater than a threshold value. Stewart and Golden [122], Laporte et al. [81] have shown that their CCP models can be transformed into an equivalent deterministic VRP model under some mild assumptions and therefore can be solved using existing deterministic VRP algorithms.

Under the reoptimization concept, after fulfilling the demand of a customer,
the vehicle makes decisions on which customer to visit next or whether a return trip is performed based on its available capacity and the set of unserved customers \cite{116}. In other words, the output of the reoptimization approach is a policy that prescribes how the route should evolve, rather than a set of preplanned routes \cite{103}. Dror et al. \cite{45} formulated the reoptimization-type VRPSD as a Markov decision process (MDP) model. However, they neither discussed the structural properties of the model nor attempted to devise solution procedures. To solve the reoptimization-type VRPSD, Secomandi \cite{117,118,115} expended significant efforts in proposing one-step rollout algorithms, which iteratively apply the cyclic heuristic of Bertsimas et al. \cite{20}. Novoa and Storer \cite{95} developed a two-step rollout algorithm that provides better solutions than the one-step version. Secomandi and Margot \cite{116} presented a finite-horizon MDP model that is more restrictive than the previous ones, and heuristically solved the model with the help of two partial reoptimization heuristics, i.e., the partitioning heuristic and the sliding heuristic. They showed by thorough numerical experiments that their proposed approaches outperform the rollout algorithms of Secomandi \cite{118,115}.

More researchers have adopted the a priori optimization solution concept to deal with the VRPSD, which is relied on a two-stage stochastic programming model. The first stage corresponds to the construction of a set of a priori routes before any customer demand is revealed, where each customer is visited exactly once. In the second stage, all customers are served in the order specified by the a priori routes, where recourse actions may be taken. The a priori route specifies the order in which customers will be served, while the actual route a vehicle would travel includes the possible recourse trips. Two recourse strategies have been considered in the a priori optimization approaches in literature \cite{116}. The first one is called reactive recourse strategy, which only requires the vehicle to take a return trip to the depot when encountering a route failure. The other
one is called *proactive recourse strategy*, which is more flexible than the reactive one. Under this recourse strategy, if the route failure occurs, the vehicle must return to the depot. Additionally, after fully serving a customer, the vehicle is allowed to either perform a recourse action (i.e., return to the depot to unload even if it is not full and then visit the next customer) or directly move to the next customer. This decision is made based on the available vehicle capacity and the set of unserved customers. Obviously, in terms of the solution quality the proactive recourse strategy dominates the reactive recourse strategy, while it requires more computational efforts.

Exact algorithms and heuristics (including meta-heuristics) have been applied to the a priori optimization version of the VPRSD. Gendreau et al. [56] proposed the first exact algorithm (i.e., the standard L-shaped algorithm) for the VRPSD with stochastic customers. Hjorring and Holt [64] applied the L-shaped algorithm to the VRPSD with a single vehicle, where a set of new cuts, called general optimality cuts, is used to enhance the performance of the algorithm. Further, Laporte et al. [84] presented an improved L-Shaped Algorithm which is capable of solving larger VRPSD instances. In this algorithm, new lower bounds at the root node are used to reduce the computation times. Christiansen and Lysgaard [29] formulated the VRPSD as a set-partitioning problem and developed a branch-and-price algorithm to solve it. The column generation subproblem is a variant of the shortest path problem, which is handled by a dynamic programming algorithm.

The exact algorithms can only solve moderate-size instances in reasonable computation times. For the large-size instances, more previous articles resorted to heuristics or meta-heuristics. The pioneering heuristic for the VRPSD is the savings algorithm of Dror and Trudeau [44], which is modified from the Clarke and Write’s savings algorithm. Later, Bertsimas [21] found closed-form
expressions and algorithms to compute the expected length of a priori route under general probability distributions. Based on these expressions, he proposed a cyclic heuristic and analyzed its worst case performance and average behavior. This cyclic heuristic was further improved by Bertsimas et al. [20], where 2-inter-exchange and dynamic programming algorithm are incorporated. Yang et al. [128] presented an Or-opt exchange algorithm for the single-vehicle VRPSD and then proposed two constructive heuristics, namely, the route-first-cluster-next heuristic and the cluster-first-route-second heuristic, for the problem with multiple vehicles. Bianchi et al. [23] implemented five basic meta-heuristics for the VRPSD, which are simulated annealing, tabu search, iterated local search, ant colony optimization and evolutionary algorithms. Rei et al. [104] designed a novel heuristic that relies on both Monte Carlo sampling and the local branching technique of Fischetti and Lodi [52], where the Monte Carlo sampling is used to approximate the recourse cost. Most of the above mentioned articles employ simple recourse policies under which the recourse cost of a certain route is not affected by other routes. Ak and Erera [4] introduced an alternative recourse policy, named paired locally coordinated operating scheme, under which a priori vehicle routes are paired and two vehicles in a pair can cooperate. Consequently, the expected recourse cost is computed based on the route pairs instead of the individual routes.

In this chapter, we concentrate on developing meta-heuristic approaches to find high-quality a priori optimization solutions for the VRPSD-TBW. When executing the a priori routes, we employ a new recourse strategy (described in Section 4.2), which is more flexible and complicated than the proactive recourse strategy. In most meta-heuristics for the VRPSD, one of the key components is the process of computing the expected cost of a given solution, which usually are computationally expensive. To alleviate the computational burden, like some
previous articles (e.g., see Bianchi et al. [23], Gendreau et al. [59]), we design several approximation schemes based on the properties of the toll functions to quickly compute the approximate expected cost of the VRPSD-TBW route. Next, the methods of evaluating a removal or insertion operation are derived using the approximation schemes. Finally, we imbed the move evaluation methods into a framework of the adaptive large neighborhood search (ALNS) [111] to solve the problem. Since no benchmark instances exist in literature, we generate a set of test instances using the real information from Chinese provinces. Our data set as well as the detailed solution results serve as benchmarks for future researchers on the VRPSD-TBW.

The remainder of the chapter is organized as follows. In Section 4.2 we formally describe the VRPSD-TBW as well as some of its properties. In Section 4.3 we introduce three approximation schemes used to compute the approximate expected cost of a route. In Section 4.4 we detail the ALNS heuristic, including the framework, the large neighborhood, the adaptive mechanism, the penalized objective function, the acceptance criteria, the generation of initial solution, the move evaluation methods, and the removal and insertion heuristics. The computational results are reported in Section 4.5 followed by the conclusion with some closing remarks in Section 4.6.

4.2 Problem description and properties

The VRPSD-TBW is defined on an undirected and complete graph $G = (V, E)$, where $V = C \cup \{0\}$ is the vertex set and $E = \{(i, j) : i, j \in V, i \neq j\}$ is the edge set. The vertex set $V$ consists of a customer set $C = \{1, 2, \ldots, n\}$ and a depot 0. The edge lengths $t_{i,j}$ are assumed to be symmetric and satisfy the triangle inequality. We are given a set of $M$ identical vehicles each with an integral weight capacity $Q$ and a curb weight $Q_c$. Each customer $i \in C$ has a discrete
and stochastic weight demand $\xi_i$ with known probability mass function $p_i(k) = \text{Prob}\{\xi_i = k\}, k = 0, 1, \ldots, K \leq Q$, which is required to be collected to the depot. The realization of $\xi_i$ becomes known upon the first arrival of the vehicle at customer $i$. All random variables $\xi_i$ are independent. The toll function $c(w)$ is monotonically increasing and composed of $H$ pieces, each of which is either linear or quadratic. Let $a_hw^2 + b_hw + c_h$ denote the $h$-th ($h = 1, \ldots, H$) piece of the toll function, where $w$ is the vehicle weight, $Q_h$ is the $h$-th breaking point, $Q_0 = 0$ and $Q_H = Q$. The toll function can be generally represented as:

$$
c(w) = \begin{cases} 
a_1w^2 + b_1w + c_1, & Q_0 \leq w \leq Q_1 \\
\vdots \\
a_hw^2 + b_hw + c_h, & Q_{h-1} \leq w \leq Q_h \\
\vdots \\
a_Hw^2 + b_Hw + c_H, & Q_{H-1} \leq w \leq Q_H
\end{cases}
$$

The objective of the VRPSD-TBW is to determine a set of $M$ a priori vehicle routes, each starting and ending at the depot, such that each customer is fully served by exactly one vehicle and the total expected transportation cost is minimized. We employ a dynamic recourse strategy when executing the a priori routes, which is more flexible and complicated than the proactive recourse strategy appearing in literature. Under the proactive recourse strategy (e.g., see Yang et al. [128], Novoa and Storer [95]), the vehicle collects as many demands from the current customer as possible. If the remaining capacity is sufficient, then after completely serving the current customer the vehicle needs to make a decision on whether moving to the next customer or performing a return tip to the depot; otherwise, the vehicle has to perform a split collection, moves to the depot and then returns to the last-visited customer. Under the dynamic recourse
strategy, upon arriving at a certain customer, the vehicle is allowed to collect any proportion of customer demand and then perform a return trip to the depot even if its remaining capacity is sufficient. In other words, there may exist multiple back and forth trips between a certain customer and the depot. It is not difficult to imagine that the dynamic recourse strategy is possible to reduce the transportation cost when some toll functions are imposed. Similar to the proactive recourse strategy, after fully serving a customer, the vehicle can also perform a return trip to the depot or move to the next customer.

Let \( r = (v_0 = 0, v_1, v_2, \ldots, v_{|r|}, v_{|r|+1} = 0) \) denote an a priori route, where \(|r|\) is the number of customers in this route. Suppose when the vehicle arrives at customer \( v_i \) (may not be the first arrival), its carried cargo weight is \( q \) and the demand (or remaining demand) of this customer is \( d \). We define \( f_i(q, d) \) as the expected cost of the partial route \((v_i, v_{i+1}, \ldots, v_{|r|}, 0)\) executed by the vehicle with a state \((v_i, q, d)\). There are three possible choices for the vehicle upon its arrival at customer \( v_i \). The first choice is to collect a proportion of demand (i.e., less than \( d \)) from customer \( v_i \), move to the depot to unload all cargoes and then return to customer \( v_i \). Under this choice, we compute the expected cost of the partial route \((v_i, v_{i+1}, \ldots, v_{|r|}, 0)\) by:

\[
g_i^1(q, d) = \min_{x \leq d \text{ and } x \leq Q-q} \left\{ c(Q_c + q + x)t_{v_i,0} + c(Q_c)t_{0,v_i} + f_i(0, d-x) \right\}, \quad (4.2)
\]

where \( x \) is a decision variable indicating the amount of the collected demand, \( c(Q_c + q + x)t_{v_i,0} \) is the cost of the back and forth trip, and \( f_i(0, d-x) \) is the expected cost of the partial route associated with the state \((v_i, 0, d-x)\).

The second choice is directly moving to the next customer \( v_{i+1} \) if \( q + d \leq Q \), where \( g_i^2(q, d) \) denotes the corresponding expected cost of the partial route \((v_i, v_{i+1}, \ldots, v_{|r|}, 0)\). By conditioning on the demand realization of customer \( v_{i+1} \),
\( g_i^2(q, d) \) can be computed as:

\[
g_i^2(q, d) = c(Q_c + q + d)t_{v_i, v_{i+1}} + \sum_k f_{i+1}(q + d, k)p_{i+1}(k)
\]

The last choice is taking all the remaining demand \( d \) at customer \( v_i \) to the depot and then directly moving to the next customer \( v_{i+1} \) if \( q + d \leq Q \). We use \( g_i^3(q, d) \) to represent the expected cost associated with this choice, which can be computed as:

\[
g_i^3(q, d) = c(Q_c + q + d)t_{v_i, 0} + c(Q_c)t_{0, v_{i+1}} + \sum_k f_{i+1}(0, k)p_{i+1}(k)
\]

Hence, the value of \( f_i(q, d) \) can be computed by the following dynamic programming recursion:

\[
f_i(q, d) = \min \begin{cases} 
g_i^1(q, d) \\
g_i^2(q, d), & \text{if } q + d \leq Q \\
g_i^3(q, d), & \text{if } q + d \leq Q 
\end{cases} \tag{4.3}
\]

with the boundary condition

\[
f_{[r]}(q, d) = \min \begin{cases} 
g_{[r]}^1(q, d) \\
g_{[r]}^2(q, d), & \text{if } q + d \leq Q 
\end{cases} \tag{4.4}
\]

Letting \( E[r] \) denote the expected cost of \( r \), we have \( E[r] = f_0(0, 0) \). Now we show the monotonicity of the function \( f_i(q, d) \), where \( 0 \leq q \leq Q \) and \( 0 \leq d \leq Q \).

**Theorem 4.1** \( f_i(q, d) \) is a monotonically increasing function of \( q \) and \( d \) for \( i = 0, \ldots, n \), \( 0 \leq q \leq Q \) and \( 0 \leq d \leq Q \).
Proof: We first show that \( f_{|r|}(g, d) \) is monotonically increasing when \( 0 \leq q \leq Q \) and \( 0 \leq d \leq Q \). Suppose \( 0 \leq q_1 \leq q_2 \leq Q, 0 \leq d_1 \leq d_2 \leq Q \) and obtaining the value of \( g_{|r|}^1(q_2, d_2) \) requires \( \eta \) back and forth trips between customer \( v_{|r|} \) and the depot. Letting \( \hat{x}_k \) be the amount of demand collected in the \( k \)-th trip, we have \( d_2 \geq \sum_{k=1}^{\eta} \hat{x}_k \) and:

\[
g_{|r|}^1(q_2, d_2) = c(Q_c + q_2 + \hat{x}_1)t_{v_{|r|}, 0} + \sum_{k=2}^{\eta} c(\hat{Q}_c + \hat{x}_k)t_{v_{|r|}, 0} \\
+ \eta c(Q_c)t_{0, v_{|r|}} + f_{|r|}(0, d_2 - \sum_{k=1}^{\eta} \hat{x}_k) \\
= c(Q_c + q_2 + \hat{x}_1)t_{v_{|r|}, 0} + \sum_{k=2}^{\eta} c(\hat{Q}_c + \hat{x}_k)t_{v_{|r|}, 0} \\
+ \eta c(Q_c)t_{0, v_{|r|}} + c(Q_c + d_2 - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_{|r|}, 0}.
\]

Since \((\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_\eta)\) is also a feasible collection pattern for state \((v_{|r|}, q_1, d_2)\) under the first choice, we can derive that:

\[
g_{|r|}^1(q_1, d_2) \leq c(Q_c + q_1 + \hat{x}_1)t_{v_{|r|}, 0} \\
+ \sum_{k=2}^{\eta} c(\hat{Q}_c + \hat{x}_k)t_{v_{|r|}, 0} + \eta c(Q_c)t_{0, v_{|r|}} + c(Q_c + d_2 - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_{|r|}, 0}.
\]

We can immediately obtain that:

\[
g_{|r|}^1(q_2, d_2) - g_{|r|}^1(q_1, d_2) \geq c(Q_c + q_2 + \hat{x}_1)t_{v_{|r|}, 0} - c(Q_c + q_1 + \hat{x}_1)t_{v_{|r|}, 0} \geq 0.
\]

If \( \sum_{k=1}^{\eta} \hat{x}_k \leq d_1 \), then \((\hat{x}_1, \ldots, \hat{x}_\eta)\) is a feasible collection pattern for state \((v_{|r|}, q_2, d_1)\). Therefore, we have:

\[
g_{|r|}^1(q_2, d_1) \leq c(Q_c + q_2 + \hat{x}_1)t_{v_{|r|}, 0} \\
+ \sum_{k=2}^{\eta} c(\hat{Q}_c + \hat{x}_k)t_{v_{|r|}, 0} + \eta c(Q_c)t_{0, v_{|r|}} + c(Q_c + d_1 - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_{|r|}, 0}.
\]
and

\[ g^1_{|r|}(q_2, d_2) - g^1_{|r|}(q_2, d_1) = \left( c(Q_c + d_2 - \sum_{k=1}^{\eta} \hat{x}_k) - c(Q_c + d_1 - \sum_{k=1}^{\eta} \hat{x}_k) \right) t_{v_{|r|}, 0} \geq 0. \]

When \( \sum_{k=1}^{\eta} \hat{x}_k > d_1 \), we can identify an integer number \( \eta' (\eta' < \eta) \), which satisfies that \( d_1 - \sum_{k=1}^{\eta'} \hat{x}_k < \hat{x}_{\eta'+1} \). Then, \((\hat{x}_1, \ldots, \hat{x}_{\eta'})\) must be a feasible collection pattern for state \( g^1_{|r|}(q_2, d_1) \). Therefore, we have:

\[ g^1_{|r|}(q_2, d_1) \leq c(Q_c + q_2 + \hat{x}_1) t_{v_{|r|}, 0} + \sum_{k=2}^{\eta'} c(Q_c + \hat{x}_k) t_{v_{|r|}, 0} + \eta' c(Q_c) t_{0, v_{|r|}} + c(Q_c + d_1 - \sum_{k=1}^{\eta} \hat{x}_k) t_{v_{|r|}, 0} \]

and

\[ g^1_{|r|}(q_2, d_2) - g^1_{|r|}(q_2, d_1) \geq \sum_{k=\eta'+2}^{\eta} c(Q_c + \hat{x}_k) t_{v_{|r|}, 0} + c(Q_c + d_2 - \sum_{k=1}^{\eta} \hat{x}_k) t_{v_{|r|}, 0} + (\eta - \eta') t_{0, v_{|r|}} + \left( c(Q_c + \hat{x}_{\eta'+1}) - c(Q_c + d_1 - \sum_{k=1}^{\eta} \hat{x}_k) \right) t_{v_{|r|}, 0} \geq 0. \]

Now we can judge that \( g^1_{|r|}(q, d) \) is monotonically increasing. Since \( g^2_{|r|}(q, d) = c(Q_c + q + d) t_{v_{|r|}, 0} \) is obviously a monotonically increasing function, according to Expression (4.4) we can conclude that \( f_{|r|}(q, d) \) is monotonically increasing.

Next we show by induction that \( f_i(q, d) \) is monotonically increasing. We will prove that if \( f_{i+1}(q, d) \) is monotonically increasing, then so is \( f_i(q, d) \). Suppose \( 0 \leq q_1 \leq q_2 \leq Q, 0 \leq d_1 \leq d_2 \leq Q \) and \( q_2 + d_2 \leq Q \). we can easily derive that \( g^2_i(q, d) \) is a monotonically increasing function since:

\[ g^2_i(q_2, d_2) - g^2_i(q_1, d_2) = \left( c(Q_c + q_2 + d_2) - c(Q_c + q_1 + d_2) \right) t_{v_i, v_{i+1}} + \sum_k \left( f_{i+1}(q_2 + d_2, k) - f_{i+1}(q_1 + d_2, k) \right) p_{i+1}(k) \geq 0. \]
and

\[ g_i^2(q_2, d_2) - g_i^2(q_2, d_1) = \left( c(Q_c + q_2 + d_2) - c(Q_c + q_2 + d_1) \right) t_{v_i,v_{i+1}} + \sum_k \left( f_{i+1}(q_2 + d_2, k) - f_{i+1}(q_2 + d_1, k) \right) p_{i+1}(k) \geq 0. \]

Similarly, we can prove that \( g_i^3(q, d) \) is also monotonically increasing. By the same process used in proving the monotonicity of \( g_i^1(q, d) \), we can obtain that \( g_i^1(q, d) \) is monotonically increasing. Hence, we can finally conclude that \( f_i(q, d) \) is a monotonically increasing function with respect to \( q \) and \( d \).

If \( c(w) \) is a linear function, \( g_i^1(q, d) \) can reduce to:

\[ g_i^1(q, d) = c(Q_c + q + \hat{x}) t_{v_i,0} + c(Q_c) t_{0,v_i} + \min \left\{ g_i^2(0, d - \hat{x}), g_i^3(0, d - \hat{x}) \right\}, \]  

where \( \hat{x} = \min\{d, Q - q\} \). Specifically, there are at most two return trips from customer \( v_i \) to the depot and the collected demand associated with the first return trip must be \( \min\{d, Q-q\} \). The correctness of the above statement can be derived from the following two theorems.

**Theorem 4.2** If \( c(w) \) is linear, \( 0 \leq d_1 \leq d_2 \leq Q \) and \( 0 \leq q_1 \leq q_2 \leq Q \), then \( f_i(q_2, d_2) - f_i(q_1, d_2) \geq (c(q_2) - c(q_1)) t_{v_i,0} \) and \( f_i(q_2, d_2) - f_i(q_2, d_1) \geq (c(d_2) - c(d_1)) t_{v_i,0} \), for \( i = 0, \ldots, |r| \).

**Proof:** When the toll function is linear, i.e., \( c(w) = bw + c (b, c \geq 0) \), the lower bound to the cost of transporting \( \delta \) units of demand from customer \( v_i \) to the depot is \( b \ast \delta \ast t_{v_i,0} \) as \( t_{v_i,0} \) is the shortest distance from customer \( v_i \) to the depot. Hence \( f_i(q_2, d_2) - f_i(q_1, d_2) \geq b \ast (q_2 - q_1) \ast t_{v_i,0} = (c(q_2) - c(q_1)) t_{v_i,0} \) and \( f_i(q_2, d_2) - f_i(q_2, d_1) \geq b \ast (d_2 - d_1) \ast t_{v_i,0} = (c(d_2) - c(d_1)) t_{v_i,0} \).

**Theorem 4.3** If \( c(w) \) is linear, then \( g_i^1(0, d) > g_i^3(0, d) \), for \( 0 \leq d \leq Q \) and \( i = 0, \ldots, |r| \).
Proof: Suppose obtaining the value of $g^1_i(0, d)$ requires $\eta$ back and forth trips between customer $v_i$ and the depot, where $\hat{x}_k$ is the amount of demand collected in the $k$-th trip. Then, we have

$$g^1_i(0, d) = \sum_{k=1}^{\eta} c(\hat{x}_k + Q_c)t_{v_i,0} + \eta c(Q_c)t_{0,v_i} \tag{4.1}$$

$$+ \min \left\{ g^2_i(0, d - \sum_{k=1}^{\eta} \hat{x}_k), g^3_i(0, d - \sum_{k=1}^{\eta} \hat{x}_k) \right\}.$$ 

Since $c(w)$ is linear, we can derive that

$$\sum_{k=1}^{\eta} c(\hat{x}_k + Q_c)t_{v_i,0} + \eta c(Q_c)t_{0,v_i} + g^2_i(0, d - \sum_{k=1}^{\eta} \hat{x}_k) \tag{4.2}$$

$$= \sum_{k=1}^{\eta} c(\hat{x}_k + Q_c)t_{v_i,0} + \eta c(Q_c)t_{0,v_i} + c(d - \sum_{k=1}^{\eta} \hat{x}_k + Q_c)t_{v_i,0} + c(Q_c)t_{0,v_i+1} + \sum_{k} f_{i+1}(0, k)p_{i+1}(k) \tag{4.3}$$

$$> c(d + Q_c)t_{v_i,0} + c(Q_c)t_{0,v_i+1} + \sum_{k} f_{i+1}(0, k)p_{i+1}(k) = g^3_i(0, d) \tag{4.4}.$$ 

Theorem 4.2 leads to the following result:

$$g^2_i(0, d - \sum_{k=1}^{\eta} \hat{x}_k) = c(d + Q_c - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_i,v_i+1} \tag{4.5}$$

$$+ \sum_{k} f_{i+1}(d - \sum_{k=1}^{\eta} \hat{x}_k, k)p_{i+1}(k) \geq c(d + Q_c - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_i,v_i+1} \tag{4.6}$$

$$+ \sum_{k} \left( f_{i+1}(0, k) + c(d + Q_c - \sum_{h=1}^{\eta} \hat{x}_h)t_{v_{i+1},0} - c(Q_c)t_{v_{i+1},0} \right) p_{i+1}(k) \tag{4.7}$$

$$= c(d + Q_c - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_i,v_i+1} \tag{4.8}$$

$$+ c(d + Q_c - \sum_{k=1}^{\eta} \hat{x}_k)t_{v_{i+1},0} - c(Q_c)t_{v_{i+1},0} + \sum_{k} f_{i+1}(0, k)p_{i+1}(k). \tag{4.9}$$
Suppose the toll function $c(w) = bw + c$, then

$$c(d + Q_c) = \sum_{k=1}^{\eta} \hat{x}_k t_{v_i,v_{i+1}} + c(d + Q_c) - \sum_{k=1}^{\eta} \hat{x}_k t_{v_{i+1},0} - c(Q_c) t_{v_{i+1},0}$$

$$= b(d - \sum_{k=1}^{\eta} \hat{x}_k)(t_{v_i,v_{i+1}} + t_{v_{i+1},0}) + (c + bQ_c) t_{v_i,v_{i+1}}$$

$$\geq \left(c(d + Q_c) - \sum_{k=1}^{\eta} \hat{x}_k\right) t_{v_i,0} + c(Q_c) t_{v_i,v_{i+1}}.$$

Therefore, we obtain

$$\sum_{k=1}^{\eta} c(\hat{x}_k + Q_c) t_{v_i,0} + \eta c(Q_c) t_{0,v_i} + g^2_i(0,d - \sum_{k=1}^{\eta} \hat{x}_k)$$

$$\geq \sum_{k=1}^{\eta} c(\hat{x}_k + Q_c) t_{v_i,0} + \eta c(Q_c) t_{0,v_i}$$

$$+ \left(c(d + Q_c) - \sum_{k=1}^{\eta} \hat{x}_k\right) t_{v_i,0} + c(Q_c) t_{v_i,v_{i+1}} + \sum_k f_i(0,k) p_{i+1}(k)$$

$$> c(d + Q_c) t_{v_i,0} + c(Q_c) t_{0,v_{i+1}} + \sum_k f_i(0,k) p_{i+1}(k) = g^3_i(0,d).$$

Finally, we can conclude that

$$g^1_i(0,d) > g^3_i(0,d).$$

Theorem 4.3 implies that there is at most one back and forth trip when computing $f_i(q,d)$ with linear $c(w)$. Theorem 4.2 tells us that $h(x) = c(q + x + Q_c) t_{v_{i+1},0} + c(Q_c) t_{0,v_i} + f_i(0,d - x)$ is a non-increasing function of $x$ so that the best collected amount of demand must be $\min\{d,Q - q\}$. Therefore, when $c(w)$ is linear, we can replace Expression (4.2) with Expression (4.5).
4.3 Approximation of the expected cost

The expected cost of an a priori route can be exactly computed using Expression (4.3). However, the computational efforts required exponentially increases as the length of the route. In this section, we propose the following three schemes to approximate the expected cost of an a priori route.

In the first approximation scheme, we transform an a priori route to a deterministic route by setting the demand realization (denoted by $d_{v_i}$) of customer $v_i$ equal to its demand expectation, namely $E[\xi_{v_i}]$. For a given an priori route, the sum of all demand expectations may be greater than the vehicle capacity. To deal with this situation, we ignore the vehicle capacity by setting $Q_H = Q = +\infty$. Thus, the expected cost of route $r$ can be approximated as:

$$d(r) = c(Q_c) t_{0,v_1} + \sum_{i=1}^{\lvert r \rvert - 1} c(Q_c + \sum_{j=1}^{i} d_{v_j}) t_{v_i,v_{i+1}} + c(Q_c + \sum_{i=1}^{\lvert r \rvert} d_{v_i}) t_{v_{\lvert r \rvert},0}$$

We call this approximation scheme the mean demand evaluation (MDE) scheme.

To compute $f_i(q,d)$, we need to decide the back and forth trips between customer $v_i$ and the depot, and the amount of demand (i.e., $\tilde{x}_k$, $1 \leq k \leq \eta$) taken in each trip. This has to be done by exhaustive enumeration, which is computationally expensive. The main idea of the second approximation scheme, called the basic heuristic evaluation (BHE) scheme, is to heuristically decide the back and forth trip and thus to simplify the recursion in the dynamic programming. The BHE scheme requires a breaking point $Q_h$ ($1 \leq h \leq H$) as input parameter and assumes that there is only one back and forth trip for a given state. The quantity collected from customer $v_i$ in the back and forth trip is determined as follows: if $q + d + Q_c \leq Q_h$, then the collected quantity $\tilde{x} = d$; if $q + d + Q_c > Q_h$ and $q + Q_c \leq Q_h$, then $\tilde{x} = Q_h - q - Q_c$; and if $q + Q_c > Q_h$, then $\tilde{x} = 0$. Under
the above strategy, \( g_1^i(q, d) \) can be rewritten as:

\[
\bar{g}_i^1(q, d) = \begin{cases} 
  c(Q_c + q + d)t_{v_i,0} + c(Q_c)t_{0,v_i} + \min\{g_2^i(0, 0), g_3^i(0, 0)\}, & \text{if } q + d + Q_c \leq Q_h \\
  c(Q_h)t_{v_i,0} + c(Q_c)t_{0,v_i} + \min\{g_2^i(0, q + d + Q_c - Q_h), g_3^i(0, q + d + Q_c - Q_h)\}, & \text{if } q + d > Q_h \text{ and } q \leq Q_h \\
  c(q + Q_c)t_{v_i,0} + c(Q_c)t_{0,v_i} + \min\{g_2^i(0, d), g_3^i(0, d)\}, & \text{if } q + Q_c > Q_h
\end{cases}
\]

We now design a new dynamic programming recursion for the BHE scheme. Define \( \bar{f}_i(q) \) as the expected cost of the partial route \((v_i, v_{i+1}, \ldots, v_{|r|}, 0)\) given that the weight of the carried cargoes is \( q \) after fully serving customer \( v_i \). With a given state \((v_i, q)\), the vehicle has two choices. The first choice is directly moving to customer \( v_{i+1} \), where the associated expected cost of the partial route can be computed by:

\[
\bar{g}_i^2(q) = \begin{cases} 
  c(q + Q_c)t_{v_i,v_{i+1}} + \sum_{k \leq Q_h - q - Q_c} \bar{f}_{i+1}(k + q)p_{i+1}(k) \\
  + \sum_{k > Q_h - q - Q_c} (c(Q_h)t_{v_{i+1},0} + c(Q_c)t_{0,v_{i+1}} \\
  + \bar{f}_{i+1}(q + k + Q_c - Q_h))p_{i+1}(k), & \text{if } q + Q_c \leq Q_h \\
  c(q + Q_c)t_{v_i,v_{i+1}} + \sum_{k} (c(q + Q_c)t_{v_{i+1},0} \\
  + c(Q_c)t_{0,v_{i+1}} + \bar{f}_{i+1}(k))p_{i+1}(k), & \text{if } q + Q_c > Q_h
\end{cases}
\]

The second choice is returning to the depot to unload all cargoes and then moving to the next customer \( v_{i+1} \). The expected cost of the partial route can be
computed as:

\[ \bar{g}_3^i(q) = c(q + Qc)t_{v_i,0} + c(Qc)t_{0,v_i+1} + \sum_{k+Qc \leq Q_h} \bar{f}_{i+1}(k)p_{i+1}(k) \]
\[ + \sum_{k+Qc > Q_h} (c(Qh)t_{v_i+1,0} + c(Qc)t_{0,v_i+1} + \bar{f}_{i+1}(k + Qc - Qh)) p_{i+1}(k) \]

Hence, \( \bar{f}_i(q) \) can be computed by:

\[ \bar{f}_i(q) = \min \left\{ \bar{g}_1^i(q), \bar{g}_3^i(q) \right\} \] (4.6)

with the boundary condition

\[ \bar{f}_i[q](q) = c(q + Qc)t_{v_i[q],0}. \]

It is easy to observe that the state and time complexities of Expression (4.6) are both \( O(nQ) \). Compared with Expression (4.3) that has a state complexity of \( O(nQ^2) \) and a time complexity of \( O(nQ^3) \), obtaining the value of \( \bar{f}_i(q) \) requires much less computational efforts.

To approximate the expected cost of a route more precisely, we can extend Expression (4.6) by searching the breaking point that yields the minimal cost, which gives rise to the third approximation scheme, called the deep heuristic evaluation (DHE) scheme. The dynamic programming recursion associated with
the DHE scheme is given as follows:

\[
\tilde{g}_1^2(q) = \begin{cases} 
\min_{h,q+Q_c \leq Q_h} \left\{ c(q + Q_c) t_{v_i,v_i+1} \\
+ \sum_{k \leq Q_h - q - Q_c} \tilde{f}_{i+1}(k + q) p_{i+1}(k) \\
+ c(Q_c) t_{0,v_i+1} + \tilde{f}_{i+1}(q + Q_c + k - Q_h) p_{i+1}(k) \right\} \\
\min_{h,q+Q_c > Q_h} \left\{ c(q + Q_c) t_{v_i,v_i+1} \\
+ \sum_{k + Q_c \leq Q_h} \tilde{f}_{i+1}(k + q) p_{i+1}(k) \\
+ \sum_{k + Q_c > Q_h} \left( c(Q_h) t_{v_i+1,0} + c(Q_c) t_{0,v_i+1} + \tilde{f}_{i+1}(k + Q_c - Q_h) \right) p_{i+1}(k) \right\} 
\end{cases}
\]

\[
\tilde{g}_3^2(q) = \min_{h} \left\{ c(q + Q_c) t_{v_i,0} + c(Q_c) t_{0,v_i+1} + \sum_{k+Q_c \leq Q_h} \tilde{f}_{i+1}(k) p_{i+1}(k) \\
+ \sum_{k+Q_c > Q_h} \left( c(Q_h) t_{v_i,0} + c(Q_c) t_{0,v_i+1} + \tilde{f}_{i+1}(k + Q_c - Q_h) \right) p_{i+1}(k) \right\}
\]

\[
\tilde{f}_i(q) = \min \left\{ \tilde{g}_1^1(q), \tilde{g}_2^2(q) \right\}
\]

with the boundary condition

\[
\tilde{f}_{i,j}(q) = c(q) t_{v_i,j,0}.
\]

### 4.4 Adaptive large neighborhood search

The adaptive large neighborhood search (ALNS) heuristic was first developed by Ropke and Pisinger [111] for the pickup and delivery VRP with time windows and then was applied to solve five simple VRP variants by Pisinger and Ropke [100]. Later, it has also successfully tackled a wide variety of complex routing and
scheduling problems, such as the capacitated arc-routing problem with stochastic demands [82], the capacitated VRPSD with time windows [85], the two-echelon VRP [83], the pollution-routing problem [38] and the service technician routing and scheduling problem [76]. We now describe the ALNS heuristic we have developed for the VRPSD-TBW.

The ALNS heuristic is essentially an iterative destroy-and-repair process. At each iteration, it destroys the incumbent solution by a removal heuristic and then constructs a new solution by an insertion heuristic. The new solution is accepted to be the next incumbent solution if it satisfies the specified acceptance criteria. To diversify the search process, we develop several removal and insertion heuristics, which are randomly chosen according to an adaptive mechanism. The probability of choosing a heuristic depends on its historical performance. The framework of our ALNS heuristic is depicted in Algorithm 10 where its main components are described in the following context of this section.

Algorithm 10 The framework of the ALNS heuristic.

1: INPUT: a feasible solution $S$;
2: The best known solution $S^* \leftarrow S$;
3: while time limit has not been reached do
4: Randomly select a removal heuristic $h_r$ and an insertion heuristic $h_i$;
5: Generate a new solution $S'$ by removing $\mu$ customers out of $S$ using $h_r$ and then re-inserting them using $h_i$;
6: if $S'$ is feasible and $c(S') < c(S^*)$ then
7: $S^* \leftarrow S'$;
8: end if
9: if accept($S'$) = true then
10: $S \leftarrow S'$
11: end if
12: Update the collected scores of $h_r$ and $h_i$;
13: if the end of the current segment is reached then
14: Update the weights $\rho_i$ of all removal and insertion heuristics;
15: Set $\pi_i = 0$ for all removal and insertion heuristics;
16: end if
17: end while
18: return $S^*$. 
4.4.1 Large neighborhood

At each iteration, \( \mu \) customers are removed from the incumbent solution by a removal heuristic and then re-inserted by an insertion heuristic. Obviously, \( \mu \) can be viewed as a parameter controlling the neighborhood size. Although larger \( \mu \) can result in larger neighborhood and therefore increase the opportunity of finding better solutions, it will consume more computational efforts. In our implementation, we set \( \mu = \lceil \frac{|V|}{2} \rceil \), where \( |V| \) is the number of vertices.

4.4.2 Adaptive mechanism

We adopt a roulette-wheel mechanism to determine which removal or insertion heuristic is chosen at each iteration. Let \( \rho_i \) denote the weight of removal (or insertion) heuristic \( i \), which is initialized to one. As in Ropke and Pisinger [111], given \( m \) heuristics, a heuristic \( i \) is selected with probability \( \rho_i / \sum_{j=1}^{m} \rho_j \). The search process is divided into a number of segments, each consisting of \( \varphi \) consecutive iterations. At the end of each segment, all weights \( \rho_i \) are adjusted based on the recorded performance of the heuristics. We define \( \pi_i \) as the total score collected by heuristic \( i \) in the current segment, which is set to zero at the beginning of each segment. The score \( \pi_i \) of a heuristic \( i \) is increased by \( \sigma_1, \sigma_2 \) or \( \sigma_3 \) depending on the new solution obtained, where

- \( \sigma_1 \): The new solution is a globally best solution.
- \( \sigma_2 \): The new solution has not been visited before and is better than the incumbent solution.
- \( \sigma_3 \): The new solution is accepted, has not been visited before and is worse than the incumbent solution.

Let \( \lambda_i \) denote the number of iterations heuristic \( i \) has been applied in the current segment. If \( \lambda_i > 0 \), then at the end of each segment \( \rho_i \) is updated as:

\[
\rho_i \leftarrow \rho_i \cdot (1 - \eta)^{\lambda_i}
\]
\[
\rho_i := \alpha \rho_i + (1 - \alpha) \frac{\pi_i}{\lambda_i}
\]
where \( \alpha \in [0, 1] \) is a controlling parameter. If \( \lambda_i = 0 \), we keep \( \rho_i \) unchanged in the next segment.

### 4.4.3 Penalized objective function

The solution in which the number of routes is not equal to \( M \) is regarded as infeasible. We use a penalty function to guide the search process between feasible and infeasible regions. For a given solution \( S \), the penalized objective value is computed as:

\[
c(S) = \sum_{r \in S} E[r] + \gamma |n(S) - M|,
\]
where \( n(S) \) is the number of routes in solution \( S \) and \( \gamma \) is a self-adjusting penalty coefficient. We initialize \( \gamma \) with the size of the vertex set \( |V| \). At each iteration, if the new solution is feasible, \( \gamma = \gamma/\beta \), where \( \beta > 1 \) is an input parameter; otherwise, \( \gamma = \gamma \times \beta \). It is noted that the value of \( E[r] \) is the expected cost of route \( r \), which is computed using the dynamic programming recursion (4.3).

### 4.4.4 Acceptance criteria

Let \( S' \) be the resultant solution generated by a removal heuristic and an insertion heuristic from the incumbent solution \( S \). This new solution is handled in a simulated annealing fashion. Specifically, if \( c(S') < c(S) \), then \( S' \) is accepted; otherwise, \( S' \) is accepted with a probability \( e^{-(c(S') - c(S))/T} \), where \( T > 0 \) is the temperature. Starting from an initial value \( T_{\text{start}} \), we multiply \( T \) by a cooling
rate $\tau \in (0, 1)$ at each iteration.

### 4.4.5 Initial solution

To generate an initial solution, we first construct $M$ routes each including only one customer and then insert the remaining customers into these routes using the basic greedy insertion heuristic (described in Section 4.4.7.4). Algorithm 11 presents the generation process of the initial solution.

**Algorithm 11 Generating an initial solution.**

1: INPUT: the customer set $C$;
2: The set of the remaining customers $C_R \leftarrow C$;
3: Construct $M$ empty routes;
4: for $i = 0$ to $M$ do
5: Select from $C_R$ the customer $v$ which is farthest to the depot;
6: Insert customer $v$ into route $i$;
7: Delete customer $v$ from $C_R$;
8: end for
9: Insert all customers in $C_R$ into the $M$ routes using the basic greedy insertion heuristic;
10: return the set of $M$ routes.

### 4.4.6 Move evaluation methods

We introduce approximate and fast methods to evaluate a move in the ALNS heuristic. A move is referred to as an operation that inserts or removes a customer. The cost of a move is defined as the difference between the expected costs of the associated routes after and before the operation. As obtaining the expected cost of a route is computationally expensive, we propose approximation methods to evaluate moves. Let $r = \{v_0 = 0, v_1, \ldots, v_{|r|}, v_{|r|+1} = 0\}$ be a route. After inserting a customer $v$ between customers $v_i$ and $v_{i+1}$ (or removing a customer $v_i$), we get a resultant route $r'$ and the cost $\Delta_{v_{i}, v}$ (or $\Delta_{v_i}$). If the ALNS heuristic
employs the MDE scheme, then

$$\Delta_{v_i,v} = d(r') - d(r)$$

which can be quickly computed. If it employs the dynamic programming recursion (4.3) to exactly evaluate a route, we compute the cost of a move using the idea from Yang et al. [128]. More precisely, let $f_i(q,d)$ denote the expected cost of the partial route $(v_i, v, v_{i+1}, \ldots, v_{|r|-1}, 0)$. Then, we have

$$\Delta_{v_i,v} = \sum_{q=0}^{Q} \sum_{d=0}^{dQ} \left( f_i(q,d) - f_i(q,d) \right) / Q^2.$$ 

The above expression computes the average cost difference of inserting customer $v$ immediately after customer $v_i$, ignoring the influence on the customers before customer $v_i$. Hence, it is an approximation method to compute the cost of a move. Similarly, if the ALNS heuristic employs the BHE or DHE scheme, then $\Delta_{v_i,v}$ can be obtained by:

$$\Delta_{v_i,v} = \sum_{q=0}^{Q} \left( \tilde{f}_i(q) - \tilde{f}_i(q) \right) / Q$$

or

$$\Delta_{v_i,v} = \sum_{q=0}^{Q} \left( \tilde{f}_i(q) - \tilde{f}_i(q) \right) / Q.$$ 

The cost $\Delta_{v_i}$ associated with a removal operation can also be computed in a similar manner.

### 4.4.7 Removal and insertion heuristics

We have developed three removal heuristics and two insertion heuristics. They are adapted from the heuristics used in some previous articles, such as Ropke and
Pisinger [111], Ribeiro and Laporte [106].

4.4.7.1 Random removal

The random removal heuristic (RRH) simply and randomly selects $\mu$ customers and then removes them from the current solution. This heuristic can help diversify the search process.

4.4.7.2 Worst removal

The worst removal heuristic (WRH) removes $\mu$ customers with the largest cost savings (i.e., $-\Delta_{v_i}$). These removed customers appear to be placed in the wrong positions as their removals lead to the most significant cost reduction. Removing these customers makes it possible to place them at other better positions, thereby improving the solution quality.

4.4.7.3 Neighbor graph removal

The purpose of the neighborhood graph removal heuristic (NGRH) is to select and remove the most critical customers by exploiting the historical information. The historical information is stored in a duplicate graph of $G = (V, E)$, where the weight $w_{i,j}$ of each edge $(i, j) \in E$ is initially set to positive infinity and then is dynamically updated along the search process. The value of $w_{i,j}$ represents the smallest average incremental cost of traversing edge $(i, j)$ among all solutions visited. Suppose we have a route $r = \{0, 1, 2, \ldots, n_r, 0\}$ in a new solution. If the ALNS heuristic employs the MDE scheme, $w_{i,j}$ is updated according to:

$$w_{i,i+1} = \min \left\{ w_{i,i+1}, c(Q_c + \sum_{j=1}^{i} d_j)l_{i,i+1} \right\}.$$
If it employs the dynamic programming recursion (4.3), \( w_{i,j} \) is updated as:

\[
w_{i,i+1} = \min \left\{ w_{i,i+1}, \sum_{q=0}^{Q} \sum_{d=0}^{Q} \frac{(f_i(q,d) - f_{i+1}(q,d))}{Q^2} \right\}.
\]

Similarly, if the ALNS heuristic employs the BHE or DHE scheme, \( w_{i,j} \) is updated as:

\[
w_{i,i+1} = \min \left\{ w_{i,i+1}, \frac{\sum_{q=0}^{Q} \left( \bar{f}_i(q) - \bar{f}_{i+1}(q) \right)}{Q} \right\}
\]

or

\[
w_{i,i+1} = \min \left\{ w_{i,i+1}, \frac{\sum_{q=0}^{Q} \left( \tilde{f}_i(q) - \tilde{f}_{i+1}(q) \right)}{Q} \right\}.
\]

The values of \( w_{i,j} \) are used to remove customers that seem to be misplaced. We first calculate a score for each customer \( i \in C \) by summing up all \( w_{i,j}, j \in V \). Then, we remove \( \mu \) customers with the highest scores \( \sum_{j \in V} w_{i,j} \).

### 4.4.7.4 Basic greedy insertion

The basic greedy insertion heuristic (BGIH) greedily inserts all removed customers into the routes. It randomly selects a removed customer \( v \), examines all possible insertion positions in the current partial solution, and then inserts this customer at the position that results in the smallest \( \Delta_{v,i} \). The above process is repeated until all removed customers are inserted. The \( \Delta_{v,i} \) of inserting a customer \( v \) is computed by the move evaluation methods described in Section 4.4.6.

### 4.4.7.5 Deep greedy insertion

The deep greedy insertion heuristic (DGIH) is the same as the BGIH except the manner of selecting customers. Every time the DGIH examines all customers and all insertion positions, and then performs the insertion associated with the
customer-position pair that results in the smallest $\Delta_{v_i,v_j}$. Obviously, this heuristic runs slower than the BGIH.

4.5 Computational experiments

So far we have four methods to evaluate the quality of a move: one is called the exact method that uses the dynamic programming recursion (4.3) and the other three are called MDE, BHE and DHE methods that uses the MDE, BHE and DHE approximation schemes. The MDE method is deterministic while the other three methods are stochastic. Combining these two types of methods conduces to increasing the diversity of the ALNS heuristic. Moreover, the computation effort required by the MDE method is small and therefore it has little influence on the total running time of the ALNS heuristic. We devised three ALNS heuristics by combining the MDE method with each of the exact, BHE and DHE methods. The first heuristic, denoted by $ALNS_1$, employs the MDE and exact methods; the second one, denoted by $ALNS_2$, employs the MDE and BHE methods; the last one, denoted by $ALNS_3$, employs the MDE and DHE methods. Note that if the toll function only contains one piece, $ALNS_2$ and $ALNS_3$ are equivalent. Since $Q_h$ is the input parameter of the BHE scheme, we can denote by $ALNS_2(h)$ the $ALNS_2$ heuristic with $Q_h$.

The random removal heuristic does not need to evaluate the cost of the move. Each of the remaining removal and insertion heuristics in each ALNS heuristic has two versions, each corresponding to one type of move evaluation method. For example, $ALNS_3$ has two versions of the WRH, one using the MDE method and the other using the DHE method. As a result, each of the three ALNS heuristics involves five removal heuristics and four insertion heuristics in total.

All algorithms were coded in Java JDK 1.6. All experiments were conducted on a Dell server equipped with an Intel Xeon E5430 CPU clocked at 2.66 GHz,
8 GB RAM and Linux operating system. Computation times reported in CPU seconds on this server.

4.5.1 Test instances

We conducted experiments on the data set generated using the real-world information of seven Chinese provinces. The toll functions implemented in these provinces are provided in Table 4.1. The column “|V|” gives the number of main cities in each province. We assume that the first city in each province is the depot and each of the rest represents a customer location. Note that in the same province the toll functions for the vehicles with different weight limits $Q_L$ may be different. For example, in Fujian province, the toll function is Expression (4.1) if $Q_L \in [0, 40]$ and is Expression (4.7) if $Q_L \in (40, +\infty)$. The column “WL interval” specifies the interval for the weight limit.

$$c(w) = \begin{cases} 
0.09w, & \text{if } w \leq 10 \\
-0.0015w^2 + 0.12w - 0.15, & \text{if } 10 \leq w \leq 40 \\
0.045w + 0.45, & \text{if } 40 \leq w \leq Q_L \\
0.09w - 0.045Q_L + 2.25, & \text{if } Q_L \leq w \leq 1.3Q_L
\end{cases}$$

(4.7)

For each province, we consider four types of vehicles, namely, the vehicles with 3, 4, 5 and 6 axles. The curb weights of these four vehicle types are 10, 13, 16 and 20, respectively. In Henan province, the weight limits of these four types of vehicles are 27, 38, 51 and 69, respectively. In the other six provinces, the corresponding weight limits are 30, 43, 56 and 70. The shortest travel distance between every pair of cities within each province is retrieved using the Google map API (available at: [http://code.google.com/apis/maps/](http://code.google.com/apis/maps/)). We use two demand probability distributions at each city, which are the discrete uniform
distribution $U[0, Q]$ and the binomial distribution $B(Q, p)$. For each instance, the value of $p$ in the binomial distribution is randomly generated from the continuous uniform distribution $U[0, 1]$. The number of available vehicles in all instances is fixed to 5.

Since there are 7 provinces, 2 probability distributions and 4 vehicle types, the total number of test instances is 56. Each instance is identified by province name, vehicle type (1, 2, 3, or 4) and probability distribution ($U$ or $B$) separated by dashes (‘-’). For example, instance Guangxi-2-$U$ represents the instance related to Guangxi province, the second type of vehicle and the discrete uniform

| Province | $|V|$ $\cdot$ WL interval | Toll function |
|----------|---------------------------|---------------|
| Qinghai  | 42 $[0, +\infty)$ | $c(w) = \begin{cases} 0.45, & \text{if } w \leq 5 \\ 0.03w + 0.3, & \text{if } 5 \leq w \leq 15 \\ 0.015w + 0.525, & \text{if } 15 \leq w \leq Q_L \\ 0.03w + 0.525 - 0.015Q_L, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Guangxi  | 118 $[0, +\infty)$ | $c(w) = \begin{cases} 0.45, & \text{if } w \leq 1.3Q_L \\ \frac{1}{2}w^2 - 1.89w + 2.25, & \text{if } 1.3Q_L \leq w \leq 2Q_L \end{cases}$ |
| Fujian   | 80 $[0, 40)$ | $c(w) = \begin{cases} -0.0015w^2 + 0.12w - 0.15, & \text{if } 10 \leq w \leq Q_L \\ 0.09w - 0.0015Q_L - 0.015Q_L, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Fujian   | 80 $[40, +\infty)$ | $c(w) = \begin{cases} -0.0015w^2 + 0.12w - 0.15, & \text{if } 10 \leq w \leq 40 \\ 0.045w + 0.45, & \text{if } 40 \leq w \leq Q_L \\ 0.09w - 0.045Q_L + 2, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Guangxi  | 84 $[0, 38)$ | $c(w) = \begin{cases} 0.08w, & \text{if } w \leq 10 \\ -0.001w^2 + 0.1w - 0.1, & \text{if } 10 \leq w \leq 50 \\ 0.04w + 0.4, & \text{if } 50 \leq w \leq 1.3Q_L \end{cases}$ |
| Guangxi  | 84 $[38, +\infty)$ | $c(w) = \begin{cases} 0.08w, & \text{if } w \leq 10 \\ 0.04w + 0.75, & \text{if } 15 \leq w \leq Q_L \\ 0.09w + 0.75 - 0.05Q_L, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Henan    | 116 $[0, +\infty)$ | $c(w) = \begin{cases} 0.08w, & \text{if } w \leq 10 \\ 0.04w + 0.75, & \text{if } 15 \leq w \leq Q_L \\ 0.09w + 0.75 - 0.05Q_L, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Hubei    | 52 $[0, 40)$ | $c(w) = \begin{cases} 0.08w, & \text{if } w \leq 10 \\ 0.08w - 0.015Q_L^2 + 0.04Q_L - 0.17, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Hubei    | 52 $[40, +\infty)$ | $c(w) = \begin{cases} 0.08w, & \text{if } w \leq 10 \\ 0.08w - 0.015Q_L^2 + 0.04Q_L - 0.17, & \text{if } Q_L \leq w \leq 1.3Q_L \end{cases}$ |
| Jiangsu  | 49 $[0, 30)$ | $c(w) = \begin{cases} 0.09w, & \text{if } w \leq 10 \\ -0.00067w^2 + 0.1w - 0.033, & \text{if } 10 \leq w \leq 40 \\ 0.07w + 0.095, & \text{if } 40 \leq w \leq 1.3Q_L \end{cases}$ |
| Jiangsu  | 49 $[30, +\infty)$ | $c(w) = \begin{cases} 0.09w, & \text{if } w \leq 10 \\ -0.00067w^2 + 0.1w - 0.033, & \text{if } 10 \leq w \leq 40 \\ 0.07w + 0.095, & \text{if } 40 \leq w \leq 1.3Q_L \end{cases}$ |

Table 4.1: Toll functions implemented by the seven Chinese Provinces.
distribution. These test instances can be divided into three classes according to the number of pieces in the toll function. The first class contains the instances that use one-piece toll functions. Although all toll functions in Table 4.1 have more than one piece, after taking the curb weight into account, the toll functions in Guangxi and Jiangsu province for the first type of vehicle can be viewed as one-piece functions. Therefore, Class 1 contains 4 test instances, which are Guangxi-1-U, Guangxi-1-B, Jiangsu-1-U and Jiangsu-1-B. Similarly, we can find that there are 36 instances with two-piece toll functions, which comprise Class 2. The third class contains the remaining 16 instances whose toll functions have three pieces. All instances are available in the online supplement to this chapter at: www.computational-logistics.org/orlib/vrpsdtbw.

4.5.2 Parameter settings

The parameters used in our ALNS algorithms were selected as $\mu = \lceil |V|/2 \rceil$, $T_{\text{start}} = 100$, $\varphi = 100$, $\sigma_1 = 100$, $\sigma_2 = 20$, $\sigma_3 = 10$, $\alpha = 0.5$, $\beta = 1.1$, $\tau = 0.999$. These settings were determined according to our preliminary experiments, the past experience in the literature and the implementation of the classical ALNS heuristic by Ropke and Pisinger [111]. In addition, we impose a time limit of 1000 seconds on each run of all ALNS heuristics.

4.5.3 Computational results

The computational results are reported in Tables 4.2 – 4.4 where for each instance we present the percentage gap between the expected cost of each ALNS heuristic and the best found expected cost of all ALNS heuristics. For example, the percentage gap in the column “$ALNS_1$” is computed as:

$$Gap = \frac{ALNS_1 \text{ expected cost} - \text{the best found expected cost}}{\text{the best found expected cost}} \times 100\%$$
The third to the last row ("Cost gap (%)") gives the average cost gap of each ALNS heuristic. We recorded the time at which an ALNS heuristic achieved the best solution for each instance and show the average times in the second to the last row ("Time to best"). For each of the $ALNS_2$ and $ALNS_3$ heuristics, we computed the expected cost of the final solution of each instance using the dynamic programming recursion (4.3) and the corresponding approximation scheme (e.g., $ALNS_3$ uses the DHE scheme). The last row ("Approximation/Exact (%)") shows the average ratios of the costs obtained by them.

<table>
<thead>
<tr>
<th>Instance</th>
<th>$ALNS_1$ (%)</th>
<th>$ALNS_3$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guangxi-1-U</td>
<td>14.40</td>
<td>0.00</td>
</tr>
<tr>
<td>Guangxi-1-B</td>
<td>21.90</td>
<td>0.00</td>
</tr>
<tr>
<td>Jiangsu-1-U</td>
<td>4.70</td>
<td>0.00</td>
</tr>
<tr>
<td>Jiangsu-1-B</td>
<td>6.68</td>
<td>0.00</td>
</tr>
<tr>
<td>Cost gap (%)</td>
<td>11.92</td>
<td>0.00</td>
</tr>
<tr>
<td>Time to best</td>
<td>444.5</td>
<td>280.7</td>
</tr>
<tr>
<td>Approximation/Exact (%)</td>
<td>—</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4.2: Computational results of Class 1 instances.

Tables 4.2–4.4 clearly show that $ALNS_3$ outperforms the other ALNS heuristics. For the Class 1 instances, $ALNS_3$ achieved all the best known solutions; the average cost gaps for the Class 2 and 3 instances are very small, only 0.22% and 0.07%, respectively. $ALNS_1$ is the slowest ALNS heuristic and performed very small number of iterations, which reduces the solution quality. The average cost gaps generated by $ALNS_1$ for the three classes of instances are 11.92%, 9.53% and 6.87%, respectively. The DHE scheme is able to quickly generate an approximate expected cost that is very close to the actual expected cost of the solution, which is revealed in the bottom-left cell of each table. $ALNS_3$ performed a large number of iterations and is able to approximate the solution cost more accurately, which we think are the main reasons of its superiority. $ALNS_2(h)$, where $h=1$, 2 or 3, performed more number of iterations than $ALNS_3$ since its approximation scheme is the fastest one. However, its performance is not better
than ALNS\textsubscript{3}, which is possibly caused by the poorer performance of the BHE scheme. From the last rows of Tables 4.3 and 4.4, we can see that the ratio of ALNS\textsubscript{2}(h) increases as h. This implies that the BHE with larger \(Q_h\) is capable of more accurately approximating the expected cost of the solution, which may explain why the performance of ALNS\textsubscript{2}(h) increases as h.

Further, we made some statistics on the invoking frequency and the computation time of the removal and insertion heuristics used by ALNS\textsubscript{3}. The results are exhibited in Figure 4.1 where the heuristics with name suffix “1” (respectively, “2”) use the DHE (respectively, MDE) scheme. In this figure, we show the percentage of the invoking times and computation time of each removal or insertion heuristic. The invoking frequencies of all removal heuristics are very close, ranging from 18.36\% to 23.83\%. However, the BGIH with the DHE scheme was
<table>
<thead>
<tr>
<th>Instance</th>
<th>$ALN_S_1$ (%)</th>
<th>$ALN_S_2(1)$ (%)</th>
<th>$ALN_S_2(2)$ (%)</th>
<th>$ALN_S_2(3)$ (%)</th>
<th>$ALN_S_3$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fujian-3-U</td>
<td>6.49</td>
<td>0.86</td>
<td>0.98</td>
<td>0.23</td>
<td>0.00</td>
</tr>
<tr>
<td>Fujian-3-B</td>
<td>12.51</td>
<td>3.91</td>
<td>1.41</td>
<td>0.06</td>
<td>0.00</td>
</tr>
<tr>
<td>Fujian-4-U</td>
<td>5.18</td>
<td>0.83</td>
<td>0.57</td>
<td>0.00</td>
<td>0.28</td>
</tr>
<tr>
<td>Fujian-4-B</td>
<td>17.25</td>
<td>3.24</td>
<td>1.69</td>
<td>0.00</td>
<td>0.32</td>
</tr>
<tr>
<td>Henan-1-U</td>
<td>4.40</td>
<td>1.94</td>
<td>1.33</td>
<td>0.00</td>
<td>0.22</td>
</tr>
<tr>
<td>Henan-1-B</td>
<td>7.32</td>
<td>3.62</td>
<td>2.60</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>Henan-2-U</td>
<td>3.40</td>
<td>0.38</td>
<td>1.27</td>
<td>0.07</td>
<td>0.00</td>
</tr>
<tr>
<td>Henan-2-B</td>
<td>5.78</td>
<td>1.79</td>
<td>2.58</td>
<td>0.00</td>
<td>0.12</td>
</tr>
<tr>
<td>Hubei-3-U</td>
<td>3.24</td>
<td>1.77</td>
<td>1.59</td>
<td>0.49</td>
<td>0.00</td>
</tr>
<tr>
<td>Hubei-3-B</td>
<td>5.44</td>
<td>2.60</td>
<td>1.85</td>
<td>0.14</td>
<td>0.00</td>
</tr>
<tr>
<td>Hubei-4-U</td>
<td>2.82</td>
<td>2.07</td>
<td>1.57</td>
<td>0.27</td>
<td>0.00</td>
</tr>
<tr>
<td>Hubei-4-B</td>
<td>10.85</td>
<td>3.43</td>
<td>2.53</td>
<td>0.11</td>
<td>0.00</td>
</tr>
<tr>
<td>Qinghai-1-U</td>
<td>5.91</td>
<td>2.19</td>
<td>4.28</td>
<td>0.00</td>
<td>0.17</td>
</tr>
<tr>
<td>Qinghai-1-B</td>
<td>4.97</td>
<td>2.79</td>
<td>1.41</td>
<td>0.32</td>
<td>0.00</td>
</tr>
<tr>
<td>Qinghai-2-U</td>
<td>6.20</td>
<td>0.78</td>
<td>2.00</td>
<td>0.22</td>
<td>0.00</td>
</tr>
<tr>
<td>Qinghai-2-B</td>
<td>8.18</td>
<td>1.45</td>
<td>1.62</td>
<td>0.24</td>
<td>0.00</td>
</tr>
<tr>
<td>Cost gap (%)</td>
<td>6.87</td>
<td>2.84</td>
<td>1.83</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>Time to best</td>
<td>289.2</td>
<td>394.0</td>
<td>481.1</td>
<td>450.7</td>
<td>483.7</td>
</tr>
<tr>
<td>Exact/Approximation (%)</td>
<td>74.81</td>
<td>90.24</td>
<td>99.87</td>
<td>99.90</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Computational results of Class 3 instances.

invoked twice as many times as the DGIH with the MDE scheme. We can also easily observe that the heuristics with the MDE scheme consumed much less computation time.
Figure 4.1: (a) Removal heuristics. (b) Insertion heuristics.
4.6 Conclusions

In this chapter, we addressed a new VRP variant that involves stochastic demands and toll-by-weight scheme (VRPSD-TBW). This problem arises in Chinese expressway systems where a vehicle is charged based on its weight and travel distance and the customer demand is not known until the first arrival of the vehicle. We seek a priori optimization solutions for this problem by a dynamic programming model. A dynamic programming recursion equipped with a novel and more dynamic recourse strategy was proposed to compute the expected cost of an a priori route. To quickly evaluate a removal or insertion operation, we introduced three schemes, namely the mean demand evaluation (MDE) scheme, the basic heuristic evaluation (BHE) scheme and the deep heuristic evaluation (DHE) scheme, to approximate the expected cost of an a priori route (or partial route). We implemented several adaptive large neighborhood search (ALNS) heuristics for the problem, each of which evaluates the removal or insertion operations by employing the MDE scheme and one of the dynamic programming recursion (4.3), the BHE scheme and the DHE scheme. By experiments, we find that the ALNS heuristic equipped with the MDE and DHE schemes outperforms the other ALNS heuristics. Since this article is a pioneering work on the VRPSD-TBW, our test instances and computational results can serve as benchmarks for future researchers.
Chapter 5

Conclusions

5.1 Summary

The vehicle routing problems with cumulative cost structure, as generalizations of the classic vehicle routing problem (VRP), have a wide range of applications, arising in the procurement of humanitarian aid in the context of natural disasters, customer centric or service-based vehicle scheduling, Chinese expressways with toll-by-weight scheme and etc. Due to the different cost structure, the existing approaches for the classic VRPs cannot be applied to solve the VRPs with cumulative cost structure effectively. Designing effective algorithms for the VRPs with cumulative cost structure requires additional effort to handle the cumulative cost structure. In this thesis, we study three VRPs with cumulative cost stricture, analyze their mathematical models, derive some useful properties and propose ad hoc algorithms to tackle them. Numerical results demonstrate the effectiveness and efficiency of our approaches.

The first problem investigated in this thesis is the multiple traveling repairmen problem with distance constraints (MTRPD) which aims to design a set of routes for the repairmen to visit a set of customers so as to minimize the total arrival time at the customers while satisfying the maximum travel distance constraints. We propose two mix integer programming models to formulate this
problem: a compact arc-flow model and a set-covering model with an exponential of variables. Based on the set-covering model, we propose two branch-and-price-and-cut algorithms with respect to two types of label-setting algorithms applied to the pricing subproblem. Experiments show that the branch-and-price-and-cut algorithm that includes the bounded bi-directional label-setting algorithm and space state relaxation outperforms the other one, and is able to solve most of the test instances optimally. Our computational results serve as benchmarks for the future researchers on this problem.

Next we study the split-collection vehicle routing problem with time windows and linear weight-related cost (SCVRPTWL), which generalizes the classic split-delivery vehicle routing problem with time windows (SDVRPTW) by defining the travel cost of vehicle per unit distance as a linear function of vehicle weight. Like the MTRPD, we formulate the SCVRPTWL into an arc-flow model and a set-covering model. Also we propose an ad hoc branch-and-price-and-cut algorithm based on the set-covering model, where the pricing subproblem is a resource-constrained elementary least-cost path problem. We first prove that at least an optimal solution to the pricing subproblem is associated with an extreme collection pattern, and then design a tailored label-setting algorithm to solve it. Extensive computational experiments show that our approach can solve the SCVRPTWL every effectively and outperforms one existing exact algorithm for the SDVRPTW.

The third problem we study is the vehicle routing problem with stochastic demands and the toll-by-weight scheme (VRPSD-TBW), which considers the stochastic demands and the toll-by-weight scheme simultaneously in the context of vehicle routing. The problem calls for the design of a set of collection routes with minimal expected cost. We first derive the dynamic recursion to compute the expected cost of a route. Since the complexity of the exact dynamic recursion is
too high, it is very unlikely to be employed in a heuristic where it may be invoked thousands of times. Hence we design several fast approximation schemes and based on them propose an adaptive large neighborhood search (ALNS) algorithm. To evaluate the performance, we design a set of benchmark instances based on the real data from Chinese expressways. Computational results demonstrate the effectiveness of our approach.

5.2 Future Work

In this thesis, we design branch-and-price-and-cut algorithms for the multiple traveling repairmen problem with distance constraints (MTRPD) and the split-collection vehicle routing problem with time windows and linear weight-related cost (SCVRPTWL). However, these exact algorithms can solve only small or medium size instances. Two directions can be considered for future research, namely to improve current exact algorithms to solve large instances and to design effective heuristics for the two problems.

The performance of branch-and-price-and-cut algorithms is heavily influenced by the quality of lower bounds provided by the LP relaxation of the set-covering model. So one way to improve the performance of the exact approaches in this thesis is to incorporate more valid inequalities into the set-covering models to strengthen the lower bounds. In the last several decades, many valid inequalities have been proposed for the traveling salesman problem (TSP) and the VRPs, including the subset row inequalities [72], comb inequalities [61], clique inequalities [121] and etc. These inequalities are also valid for the MTRPD and the SCVRPTWL, and expected to improve the lower bounds. However, there exists one major difficulty which prevents them employed in the current versions of the branch-and-price-and-cut algorithms. These inequalities would destroy the structure of the current pricing problems, which makes the pricing problems very
difficult to solve. Hence our future can focus on how to incorporate these valid inequalities into current branch-and-price-and-cut algorithms.

The effectiveness of a heuristic depends on two key components heavily: the solution representation and the neighborhood operators, especially when the problem is very complicated. These two components decide the potential solutions which the heuristic can explore and the efficiency. A good solution representation and well-designed neighborhood operators enable the heuristic to explore more solutions so that it is more likely to find good solutions, and accelerate the speed of the heuristic moving from one solution to another. Hence when we are devising the heuristics for the MTRPD and the SCVRPTWL, we will mainly focus on the design of the solution representation and the neighborhood operators, which requires careful study on the problems.
Bibliography


[64] Curt Hjorring and John Holt. New optimality cuts for a single-vehicle


