12-10-2005

A Study On The Split Delivery Vehicle Routing Problem

Kai Liu

A STUDY ON THE SPLIT DELIVERY VEHICLE ROUTING PROBLEM

By

Kai Liu

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Industrial Engineering
in the Department of Industrial Engineering

Mississippi State, Mississippi

December 2005

A STUDY ON THE SPLIT DELIVERY VEHICLE ROUTING PROBLEM

By

Kai Liu

Approved:

Mingzhou Jin **Assistant Professor of Department of Industrial Engineering** **(Director of Dissertation)**	**Royce O. Bowden** **Professor and Head of Department of Industrial Engineering** **(Committee Member)**
Stanley F. Bullington **Professor and Graduate Coordinator of Department of Industrial Engineering** **(Committee Member)**	**Burak Eksioglu** **Assistant Professor of Department of Industrial Engineering** **(Committee Member)**
Kirk Schulz **Dean of the College of Engineering**	**Murat Erkoc** **Assistant Professor of Department of Industrial Engineering** **University of Miami** **(Committee Member)**

Name: Kai Liu

Date of Degree: December 10, 2005

Institution: Mississippi State University

Major Field: Industrial Engineering

Major Professor: Dr. Mingzhou Jin

Title of Study: A STUDY ON THE SPLIT DELIVERY VEHICLE ROUTING PROBLEM

Pages in Study: 85

Candidate for the Degree of Doctor of Philosophy

This dissertation examines the Split Delivery Vehicle Routing Problem (SDVRP), a relaxed version of classical capacitated vehicle routing problem (CVRP) in which the demand of any client can be split among the vehicles that visit it.

We study both scenarios of the SDVRP in this dissertation. For the SDVRP with a fixed number of the vehicles, we provide a Two-Stage algorithm. This approach is a cutting-plane based exact method called Two-Stage algorithm in which the SDVRP is decomposed into two stages of clustering and routing. At the first stage, an assignment problem is solved to obtain some clusters that cover all demand points and get the lower bound for the whole problem; at the second stage, the minimal travel distance of each cluster is calculated as a traditional Traveling Salesman Problem (TSP), and the upper bound is obtained. Adding the information obtained from the second stage as new cuts into the first stage, we solve the first one again. This

procedure stops when there are no new cuts to be created from the second stage. Several valid inequalities have been developed for the first stage to increase the computational speed. A valid inequality is developed to completely solve the problem caused by the index of vehicles. Another strong valid inequality is created to provide a valid distance lower bound for each set of demand points. This algorithm can significantly outperform other exact approaches for the SDVRP in the literature.

If the number of the vehicles in the SDVRP is a variable, we present a column generation based branch and price algorithm. First, a restricted master problem (RMP) is presented, which is composed of a finite set of feasible routes. Solving the linear relaxation of the RMP, values of dual variables are thus obtained and passed to the sub-problem, the pricing problem, to generate a new column to enter the base of the RMP and solve the new RMP again. This procedure repeats until the objective function value of the pricing problem is greater than or equal to zero (for minimum problem). In order to get the integer feasible (optimal) solution, a branch and bound algorithm is then performed. Since after branching, it is not guaranteed that the possible favorable column will appear in the master problem. Therefore, the column generation is performed again in each node after branching. The computational results indicate this approach is promising in solving the SDVRP in which the number of the vehicles is not fixed.

DEDICATION

I would like to dedicate this research to my parents, and all my friends.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Mingzhou Jin, my dissertation director, for his magnanimity in expending time and effort to guide and assist me throughout the dissertation progress. Expressed appreciation is also due to the other members of my dissertation committee, namely, Dr. Royce O. Bowden, Dr. Stanley F. Bullington, Dr. Burak Eksioglu and Dr. Murat Erkoc, for the invaluable aid and direction provided by them. Finally, I would like to thank my family and my girlfriend, Miss Zheng Gu. Their encouragement and support are great power to make me to accomplish this dissertation.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This chapter consists of four sections. In Section 1.1, we introduce the definition of the Vehicle Routing Problem (VRP) and the Split Delivery Vehicle Routing Problem (SDVRP). In Section 1.2, we illustrate the significance and objective of this research. In Section 1.3, we present the methodology of this study. Finally, we propose the organization of this dissertation in Section 1.4.

1.1 Introduction

The Vehicle Routing Problem (VRP) is a famous problem in the field of combinatorial optimization. It is defined on a graph characterized by G=(V, E), where $V = \{0, 1,\ldots, N\}$ is a set of vertices corresponding to locations, such as cities, suppliers, customers, etc., and $E = \{(i, j) : i, j \in V, i \neq j\}$ is the edge set. Vertex *0* represents a depot at which a fleet of m vehicles are based. Generally, m can be a fixed number or a variable that is defined on an interval $[\underline{m}, \bar{m}]$, where $1 \le \underline{m} \le \bar{m} \le N$, and vehicles may have equal or different capacities. In this dissertation, the vehicles are assumed to have a same capacity of Q. Every vertex i of $V\backslash\{0\}$ has a positive demand $d_i \le Q$, and every edge (i, j) has a positive distance or travel cost c_{ij}. The VRP tries to minimize the total cost with a set of vehicle routes. The routes should

satisfy the following conditions:

(1) all vehicles should start and end at the depot;

(2) every demand point is visited exactly once; and

(3) the total demand of any route does not exceed the capacity of the vehicle assigned to the route.

The VRP is known to be NP-hard [1], and there is abundant literature on the VRP and related topics [2, 3, 4, 5, 6, 7]. In this dissertation, we propose to study the Split Delivery Vehicle Routing Problem (SDVRP), which is introduced by Dror and Trudeau **[8, 9].** The SDVRP is a relaxation of the VRP without condition (2). In other words, the demand of a point can be split among several vehicles. Furthermore, the assumption of $d_i \leq Q$ is not necessary for the SDVRP. Dror and Trudeau [8] demonstrate allowing split delivery can result in significant savings both in the total travel distance and the number of used vehicles. In general, when a customer demand point's demand exceeds 10% of the vehicle capacity, the cost of the optimal solution for an SDVRP is considerably lower than that of the optimal solution for its corresponding VRP. The SDVRP is still NP-hard [8]. Figure 1.1 and 1.2 illustrate examples for the VRP and the SDVRP.

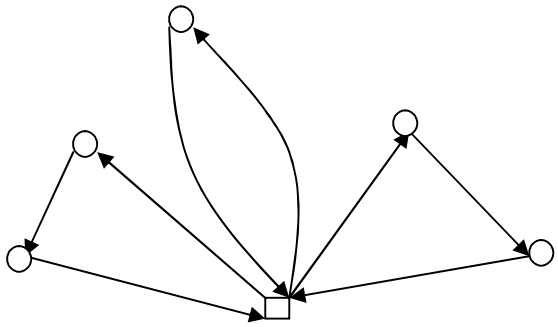

Figure 1.1: An example of VRP routes

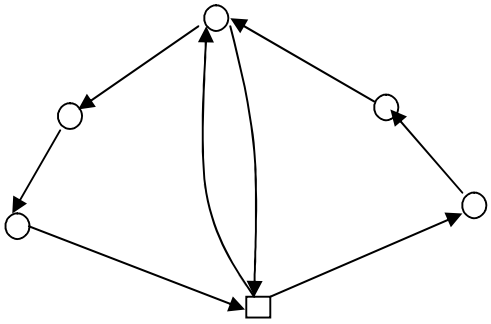

Figure 1.2: An example of SDVRP routes.

Various mathematical formulations of the SDVRP exist in the literature. Dror and Trudeau **[**8**]** present the following model:

Notation:

C_{ij} : The distance ("cost") between demand points i and j.

d_i : the demand at point i.

Q_k : The capacity of the kth vehicle.

x_{ij}^k : 1 if the kth vehicle travels directly from point i to j; 0 otherwise.

y_{ik} : The fraction of the ith point demand delivered by the kth vehicle.

U : The number of vehicles in the fleet.

S : The set of all cycles on the set V which include the depot.

$$P1: \quad \min \quad z = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{v=1}^{U} C_{ij}x_{ij}^k$$

s.t.

$$\sum_{k=1}^{U}\sum_{i=0}^{N} x_{ij}^k \geq 1, \qquad j = 0,...,N \qquad (1\text{-}1)$$

$$\sum_{i=0}^{N} x_{ip}^k - \sum_{j=0}^{N} x_{pi}^k = 0 \qquad p = 0,...,N; k = 1,...,U \qquad (1\text{-}2)$$

$$\sum_{k=1}^{U} y_{ik} = 1 \qquad i = 1,...,N \qquad (1\text{-}3)$$

$$\sum_{i=1}^{N} d_i y_{ik} \leq Q_k \qquad k = 1,...,U \qquad (1\text{-}4)$$

$$y_{ik} \leq \sum_{j=0}^{N} x_{ji}^k \qquad i = 1,...,N; k = 1,...,U \qquad (1\text{-}5)$$

$$X \in S \qquad (1\text{-}6)$$

Constraints (1-1) guarantee that each demand point is at least visited once. Constraints (1-2) are the flow conservation constraints. Constraints (1-3) insure that each point will receive its full demand. Constraints (1-4) are vehicle capacity constraints. Constraints (1-5) enforce that demand point i can be serviced only by a vehicle visiting it. The final constraints (1-6) are general sub-tour elimination constraints.

We make the following assumptions for the study:

1) The distances are symmetric, i.e., $C_{ij} = C_{ji}$ for all i, j, and satisfy the basic triangular inequality.
2) The vehicles are identical with the same capacity of Q.
3) The number of the vehicles in the fleet is sufficient to satisfy the total demand of the clients.

1.2 Objective and significance of the study

This research will focus on (1) developing a new exact method for the split delivery vehicle routing problem; and (2) applying the branch -and-price approach to obtain a good feasible integer (optimal) solution to the SDVRP. A limited-search-tree-with-bound algorithm is developed to solve the sub-problem of the column generation based formulation of the SDVRP.

Though plenty of papers have made contribution to solving the SDVRP, the research on the SDVRP is significantly behind that on the VRP. The existing algorithms cannot even solve medium-size problems well. The proposed research tries to develop new methodologies to solve the SDVRP. Based on the numerical experiments, these two proposed approaches are computationally competitive to the existing algorithms.

1.3 Research Methodology

The following steps are proposed to accomplish the objectives of this research:

1) *Model construction*: Two mathematical programming formulations of the SDVRP are presented in this dissertation. The first formulation assumes the number of used vehicles is fixed, while the second one relaxes this assumption. Different models have a large impact on the algorithm development.

2) *Algorithm development*: An algorithm can be defined as a precise rule (or a set of rules) specifying how to solve a problem. Modern computation depends heavily on computer tools (hardware and software) to solve large and complicated problems. Algorithms are developed to provide computers instructions to solve the problem step by step. Both the computational time and the solution quality are critical in algorithm development. Sometimes, some tradeoff must be made.

3) *Data generation*: Testing data can be collected from the practice or be generated randomly. In this dissertation, all data are borrowed from published papers in which the SDVRP data are generated randomly in order to make the numerical experiment result comparable.

4) *Coding*: In this dissertation, all algorithms are realized in C. The callable library of CPLEX 9.0 is used to solve linear programming models and simple sub-problems.

5) *Result comparison and analysis*: Numerical experiment results will be compared to the published papers regarding the solution quality and the computational time. Examples with optimal solutions in the published papers can help verify the proposed models and algorithms. Only computational speed is the

concern for these examples.

1.4 Organization of Dissertation

The structure of this dissertation is as follows. Chapter II introduces the literature review on the Split Delivery Vehicle Routing Problem and the column generation technique. Chapter III presents a Two-Stage exact approach to the SDVRP with efficient valid inequalities. In Chapter IV, we propose a column generation based branch-and-price method to the SDVRP when the number of vehicles in the fleet is a variable. Chapter V states the conclusion of this research and possible future extension.

CHAPTER II

LITERATURE REVIEW

This chapter includes two sections. In Section 2.1, we provide the literature review on the Split Delivery Vehicle Routing Problem with some variation. In Section 2.2, we present the literature review on the column generation technique and the branch-and-price method.

2.1 Review on the Split Delivery Vehicle Routing Problem

Dror and Trudeau introduce the SDVRP [8], where they relax one of the conditions of the Vehicle Routing Problem (VRP) and allow more than one vehicle to visit one demand point. They claim that allowing split delivery can result in significant savings both in the total travel distance and the number of vehicles required. In general, when a customer's demand exceeds 10% of the vehicle capacity, the cost of the optimal solution for an SDVRP is considerably lower than that of the optimal solution for its corresponding VRP.

Since then, the SDVRP has received more attention for the last decade both in theoretical analysis and practical application. The theoretical work includes the concept development and the optimality property analysis for the SDVRP [1, 8, 9].

Dror and Trudeau first present the concept of the Split Delivery Vehicle Routing Problem, and propose algorithms to solve this problem. They also develop some valid efficient inequalities based on their formulation of the SDVRP, and study the properties of optimal solution for the Split Delivery Vehicle Routing Problem.

2.1.1 Properties of the optimal SDVRP solutions

Theorem 2.1: If the $\{C_{ij}\}$ matrix satisfy the triangular inequality then no two routes in the optimal solution of the SDVRP can have more than one split demand point in common.

Definition 2.1: Given k demand points $v_1, v_2, ..., v_k$ and k routes. Route 1 includes the points v_1, v_2; route 2 includes points v_2, v_3; … ; route k-1 includes points v_{k-1}, v_k, and route k includes points v_k, v_1 (this implies that the points v_1, v_2,…, v_k receive split deliveries by the k respective routes and other routes as possible). This subset of demand points $\{v_i\}$ (i=1,…,k) is called a k-split cycle.

Thus, a generalization of Theorem 2.1 can be presented as follows:

Lemma 2.1: if the $\{C_{ij}\}$ matrix satisfies the triangular inequality then there is no k-split cycle (for any k) in the optimal solution to problem.

2.1.2 Formulations and algorithms for the SDVRP

In the literature, several formulations and algorithms for the Split Delivery Vehicle Routing Problem are proposed. Dror and Trudeau present an integer linear programming formulation including new families of valid inequalities, as well as an

exact constraint relaxation algorithm for the SDVRP. The formulation is given in Chapter I, and here we restate the valid inequalities and the algorithm without proof.

Proposition 2.1 (Sub-tour elimination inequalities)

$$\text{The constraints } \sum_{k=1}^{U}\sum_{i,j\in S} x_{ijk} \le \sum_{i\in S} d_i - V(S) \quad (S \subseteq N \setminus \{0\}; |S| \ge 2) \tag{2-1}$$

are equivalent to constrains (1-6) and are therefore valid inequalities for the SDVRP.

Proposition 2.2 If $C = \{c_{ij}\}$ satisfies the triangle inequality, the constraints

$$\sum_{k=1}^{U}\sum_{i,j\in S} x_{ij}^{k} \le |S| - 1 \quad (S \subseteq N \setminus \{0\}; |S| \ge 2) \tag{2-2}$$

are valid inequalities for the SDVRP.

Proposition 2.3 There always exists an optimal SDVRP solution in which the number of positive x_{ij}^{k} variables is at most equal to n+2m-1. (In the case of strict triangle inequality, the number of positive variables is at most n+2m-1 in any optimal solution.)

Proposition 2.4 (Variable fixing) When all vehicles have the same capacity, it is valid to have the following constraint:

$$\sum_{j=0} x_{i^* j1} = 1 \tag{2-3}$$

Proposition 2.5 (Fractional cycle elimination constraints I) The constraints

$$\sum_{i\in S, j\in \bar{S}} x_{ij}^{k} \ge \left(\sum_{i,j\in S} x_{ij}^{k}\right)/(|S| - 1) \quad (S \subseteq N \setminus \{0\}; |S| \ge 2; k = 1,...,U) \tag{2-4}$$

are valid inequalities for the SDVRP.

Proposition 2.6 (Fractional cycle elimination constraints II) The constraints

$$x_{ij}^{k} \le \sum_{l\ne i} x_{lj}^{k} \quad (i, j \in N \setminus \{0\}; k = 1,...,U) \tag{2-5}$$

are valid inequalities for the SDVRP.

The scheme of the algorithm is that: using heuristics to obtain one upper bound of the problem, and solving the LP relaxation of the problem with the valid inequalities except the sub-tour elimination constraints to attain the lower bound. If the solution to the lower bound is feasible, then the optimum is reached. If it is infeasible, we check for the constraint violations. If some violations are identified, we introduce a subset of all violated constraints to the original LP relaxation of the problem, and solve it again. When no violated constraints are identified, the optimum of the relaxation has been reached. Therefore, we turn to the procedure of branch and bound to obtain the optimal integer solution.

Since the authors mainly focus on the efficiency of the valid inequality for the LP relaxation problem, the instances of the SDVRP provided by them are not solved completely. Thus, the computational experiments only display the results of the root of the search tree. They claim that the various constraints developed for this problem are quite successful in reducing the gap between the lower and upper bounds at the root of the search according to the computational results.

Belenguer, Martinez and Mota provide a different formulation from Dror's since the number of vehicle in the fleet of the SDVRP they study is fixed [**10**]. They conduct research of the polyhedral property in their paper, and develop some valid inequalities for their cutting-plane algorithm as well. The formulation proposed by them is as follows:

$$
\begin{aligned}
\text{Min} \quad & \sum_{i,j\in E} c_{ij}x_{ij} \\
\text{s.t.} \quad & x(\delta(0)) \geq 2K \quad \text{and even}, \\
& x(\delta(i)) \geq 2 \quad \text{and even}, \ \forall i \in V \setminus \{0\}, \\
& x(\delta(S)) \geq 2\left\lceil \frac{d(S)}{Q} \right\rceil \ \begin{cases} \forall S \subseteq V \setminus \{0\} \\ 2 \leq |S| \leq n-1, \end{cases} \\
& x_{ij} \geq 0 \text{ and integer} \quad \forall (i,j) \in E
\end{aligned} \tag{2-7}
$$

x_{ij} : the number of times that edge (i, j) is used in a feasible solution to the SDVRP.

K : the number of the vehicles in the fleet, equals to $\lceil d(V)/Q \rceil$.

E : the set of edges in the Graph.

$d(V)$: total demand in V.

They prove that every incidence vector of a feasible solution to the SDVRP satisfied the above formulation, R_{SDVRP}, but the reciprocal is not true. Therefore, they develop a cutting-plane algorithm to obtain the optimal solution to the R_{SDVRP}, which is the lower bound of the corresponding SDVRP. The principle of the algorithm is similar as that presented by Dror and Trudeau. The detail of the algorithm is listed below:

Step 1: Init. Let j:=0 and let PL_j be the following linear problem:

$$
\begin{aligned}
& Min \ \ c^t x \\
\text{s.t.} \quad & x(\delta(0)) \geq 2\left\lceil \frac{d(V)}{Q} \right\rceil, \\
& x(\delta(i)) \geq 2, \ \forall i = 1,...,n, \\
& x_e \geq 0, \quad \forall e \in E.
\end{aligned} \tag{2-8}
$$

Step 2: Solving PL_j. Solving problem PL_j using a linear programming code. Let x^j be the optimal solution.

Step 3: Identification of violated constraints.

Step 3.1 If any violated constraint, among those in R_{SDVRP}, can be found

on G(x^j), the graph induced by the edges such that x^j>0, go to Step 4.

Step 3.2 If no violated generalized capacity constraints can be found on G(x^j), STOP.

Step 4: Updating PL_j. Add to the set of constraints of PL_j the constraints found in the previous step. Let PL_{j+1} be the resulting problem and let j:=j+1. Go to Step 2.

The authors use five procedures to identify the violated constraints. First three procedures are heuristic algorithms to identify the sub-tour, connected components and capacity constraints respectively. When they all fail to find a violated constraint, exact algorithms of identification are applied in procedure four and five.

Their method obtains good lower bounds and even optimal solutions to some instances. However, it cannot guarantee to obtain an optimal solution even with an infinite amount of time.

Lee et al. develop a dynamic programming (DP) algorithm for the SDVRP [11]. The DP has an infinite number of states and actions. They show that there is an equivalent finite action spaces DP for any given initial condition. They use a best-first shortest path search procedure in the direct network associated with the finite state DP to solve the SDVRP.

Their dynamic programming based formulation for the SDVRP is as follows:

N: the number of demand point;

d_i: the demand at the demand point $i, i = 1,...,N$;

$d = (d_1,...,d_N) \in \mathbf{R}_+^N$: the demand vector;

$r \in \{0,1\}^N$: the route in the solution.
$r_i = 1$ if and only if the vehicle following this route visits the ith demand point.

$c(r)$: total cost of executing route r.

R: the set of all feasible routes.

$z: \mathbf{R}_+^N \to \mathbf{R}_+$: the function mapping each demand vector $d \in \mathbf{R}_+^N$ to the cost of optimal routing, $z(d)$.

(E, I): decison space, where $E \subseteq \{1,...,N\}$ is the set of demand point visited and emptied, and I is the set of demand point visited, but not emptied, by the vehicle.

$w_j \in \mathbf{R}_+^N$: load vector for vehicle j.

$r(w) \in \{0,1\}^N$ such that $r_i(w) = 1(= 0)$ if and only if $w_i > 0(= 0)$

The finite action space for a given state n is:

$$A(n) = \{(E,I): n_E = 1, I = \phi \text{ or } |I| = 1, n_{E \cup I} > 1, r(E,I) \in R\} \tag{2-9}$$

The recursive equation for the DP is:

$$\begin{aligned} &z(n) = \min_{(E,I) \cup A(n)} \{c(E,I) + z(n'(n,(E,I)))\},\ n \in Z_+^N, n \neq 0 \\ &z(0) = 0 \end{aligned} \tag{2-10}$$

The state n' is a successor of state n if and only if the state n' can be attained by executing a feasible action at state n.

They choose a forward-search shortest path algorithm to solve the DP problem, since this approach can avoid considering states that are not reachable from the initial state $n(d)$. The algorithm utilizes a guidance function $f(\cdot)$ to select which of the nodes generated to explore at the next step of the search and hence to direct the search to the most promising alternative to find a good solution in its early stage.

The definition function for node *n* is:

$$f(n) = g(n) + h(n) \tag{2-11}$$

where $g(n)$ is the best currently known path from the start node s to the node *n*, with $g(s)=0$, and $h(n)$ is an estimate of the cost of the optimal path from the candidate node *n* to the destination with $h(t)=0$. The set of all nodes that have been generated but not yet explored is referred to as "OPEN", and the set of nodes have been expanded as "CLOSED".

The outline of the algorithm is as follows:

1. Put the start node *s* into OPEN; set $g(s)=0$;
2. If OPEN is empty, exit with failure.
3. Remove from OPEN a node *n* for which *f* is minimized, and place it in CLOSED.
4. If *n* is the end node, exit successfully with the solution obtained by tracing back the pointers from *n* to *s*.
5. Otherwise expand *n*, generating all its successors, and attach to them pointers back to *n*. For every successor *n*' of *n*:
   (a) If *n*' is not already in OPEN or CLOSED, compute the estimate $h(n')$, and calculate $f(n')=g(n')+h(n')$ with $g(n')=g(n)+c(n,n')$, where $c(n,n')$ is the cost of the arc from *n* to *n*'.
   (b) If n' is already in OPEN or CLOSED, direct its pointers along the path yielding the lowest $g(n')$.

(c) If *n*' required pointer adjustment and was found in CLOSED, reopen it.

6. Go to Step 2.

The main goal of Lee et al.'s work is to provide the basic idea of the a DP-based approach for solving the SDVRP to optimality, and its main contribution is the theoretical foundation of this approach, since current implementation of the algorithm is unable to handle realistic, large instances of the SDVRP. (The largest problem that they solved in a reasonable amount of time has 9 demand points and 6 vehicles.)

Frizzell and Giffin study an extension of the Split Delivery Vehicle Routing Problem where customers may have a time windows for their delivery [12, 13]. They develop a construction heuristic that uses a look-ahead approach to solve the SDVRP with time windows. The main objective of the construction heuristic is to minimize total time taken, with the possibility of a relatively large number of customers receiving split deliveries. In order to improve the performance of the heuristic, two other heuristics are applied as well. One attempts to move customers within routes, while the other exchanges customers between routes.

2.1.3 Applications on the Split Delivery Vehicle Routing Problem

In the application of the SDVRP, Mullaseril et al. use a heuristic algorithm for a livestock feed distribution problem encountered on a cattle ranch in Arizona [14]. The problem is a collection of split-delivery capacitated rural postman problem with

time windows on arcs, and is described as follows:

The livestock ranch is represented as a connected mixed graph $G=(V,A)$ where the set of edges and arcs A corresponds to road segments in front of the pens(used for delivery feed) and service road segments(for non-delivery travel), and the nodes V represent intersection/turning points in the service roads or boundaries between adjacent pens. For each type of feed, there is a subset of arcs R that requires traversal, corresponding to the pens that require delivery of that particular feed. The required set of arcs R is directed because of the design of the delivery trucks. The arcs and edges representing service roads may be undirected, allowing two-way traffic, or directed one-way traffic only. Other direct arcs may represent the alleys in front of the rows of pens when traversed in the opposite direction to feed delivery.

When there is a non-negative demand associated with the required arcs R, and a upper bound on the sum of demands delivered on a route (that is, a cycle in the graph containing the depot node), the problem of finding collection of routes that covers the demand on the required arcs R and meets the capacity bounds for each route is called a Capacitated Rural Postman Problem (CRPP).

In their study, they allow each required arc to be serviced more than one route. The solution strategy they adopt is an adaptation of the heuristics proposed for split delivery for node routing problems explored by Dror and Trudeau. First, they generate feasible solutions for the corresponding routing problem where split deliveries are not allowed, and then apply heuristics to produce and improve split

delivery solutions.

The overall solution approach includes four modules:

(i) Generating a non-split feasible solution.

In this module, two heuristics, the extended path-scan heuristic and the modified augment-merge heuristic, are used to generate a set of feasible routes. These two heuristic algorithms are first tests on the CRPP with time windows without split delivery by Dror, Leung and Mullaseril. The first heuristic algorithm constructs a feasible route one at a time until the demands of all arcs in the set of required arcs are met. In the extended augment-merge heuristic, possible merging of routes that is feasible in both capacity and time and also results in net overall savings are searched. This process stops when no merge steps are possible.

(ii) Improving the solution by arc interchange.

The arc-swapping improvement procedure is an adaptation to that for the CRPP to include time windows and is run on all feasible solutions obtained, both with and without split deliveries.

(iii) Generating split-delivery routes by *k*-split generation.

In this module, the authors check to see whether the delivery made to an arc can be split across *k* other candidate routes in such a way that the highest savings is obtained. First, 2-split generation is analyzed,

and they generalize to k-split (k>2) candidate routes. One thing needs to be concerned in this procedure is where to insert the delivery to arc (i, j) in the sequence of required arcs that make up the routes. The authors choose the position for insertion to be the one that obtains the highest savings in distance.

(iv) Modifying the solution by route addition.

They investigate arcs whose demand is split among several routes to see if consolidating them into one new route will realize a net savings in distance traversed. A k-route addition, which means taking an arc that has a split deliveries out of the various routes it is on and creating a separate route to make this delivery, is performed. In their implementation of this procedure, k=2 or 3.

The authors test this heuristic algorithm on the data from practice and achieve improvement over 10% of total distance. They also conclude that better results are obtained without time window constraints than that with time window constraints.

Another application of the SDVRP in literature is proposed by Sierksma and Tijssen [15]. The problem they deal with is to determine a fight schedule for helicopters to off-shore platform locations for exchanging crew people employed on these platforms. The helicopters carrying new people fly from the airport to the platforms for gas production in the North Sea and leaving the platforms or return to the airport with the leaving people. The only difference between their problem and

the SDVRP is that there is a range limit for the helicopters due to the quantity of fuel they carry, and no such constraint is applied to the vehicles in the SDVRP. There are 51 platforms and 27 seats in the helicopter. The authors provide the coordinate of each platform, but they do not mention the number of people at each platform for exchanging in their paper.

They form a linear integer programming model for their problem. The following notation is used in the model:

N = the number of platforms
i = platform location index, with $i = 1,...,N$;
P_i = platform with index i;
N_F = the number of feasible helicopter flight;
f = the flight index, with $f = 1,...,N_f$;
x_f = the number of times flight f is executed;
D_i = the number of demanded crew exchange for platform P_i;
a_{if} = the number of crew exchanges on platform P_i during flight f;
d_f = the cost of executing flight f once;
C = the number of available seats, called the capacity, of the helicopters.

The model they present is as follows:

$$(FF) \qquad \min \quad \sum_{f=1}^{N_F} d_f x_f$$

$$\text{s.t.} \quad \sum_{f=1}^{N_F} a_{if} x_f = D_i \quad \text{for } i = 1,...,N, \qquad (2\text{-}12)$$

$$x_f \geq 0, \text{ and integer} \quad \text{for } f = 1,...,N_F.$$

In the present model, the decision variables correspond to feasible flights, so they do not include explicit flight feasible constraints into this model. Since N_F is generally very large, the usual Simplex Method cannot be applied on the relaxation of model (FF) in which the variables do not need to be integers. Finding an entering

column for the current basis of the finite set of feasible routes will utilize the technique called “column generation” (We will illustrate this concept in section 2.2 in detail.). The authors use the following formulation to generate the entering column:

$$\begin{aligned} \text{(CG)} \qquad & \min \ (d_f - \sum_{i=1}^{N} y_i a_i) \\ & \text{s.t.} \ \sum_{i=1}^{N} a_i \le C, \\ & 0 \le a_i \le D_i, \quad \text{for } i = 1,...,N. \\ & d_f \le R, \ d_f \text{ being the length of a shortest route of} \\ & \text{the flight } f \text{ visiting the platforms } P_i \text{ with } a_i > 0 \end{aligned} \tag{2-13}$$

Model CG is a nonlinear model, because the variable d_f is dependent on the nonzero values of a_i. In fact, d_f is defined as the total traveled distance of a shortest flight from the airport to all platforms within the route and back to the airport. In order to solve the model (CG), they distinguish the following procedures:

(1) Formulate and solve a Traveling Salesman Problem and a Knapsack Problem for a fixed platform subset *S*;

(2) Generate subsets *S* of the set of all platforms for which (1) has to be solved, and discard those subsets that cannot produce an optimal solution.

Given a subset *S* of the set of all platforms, the objective function of model (CG) is rewritten as follows:

$$c(S) = d_{f(S)} - (\max \sum_{i \in S} a_i y_i), \tag{2-14}$$

With $d_{f(S)}$ is the length of a shortest flight visiting all platforms in *S* and is

solved through procedure (1). For those variables a_i in (CG), they are obtained by solving the following Knapsack Problem (KPs):

$$(KPs) \quad \max \sum_{i \in S} y_i a_i$$
$$\text{s.t.} \sum_{i \in S} a_i \le C, \tag{2-15}$$
$$0 \le a_i \le D_i \ \text{ for } i \in S$$

They utilize classic "greedy" algorithm to solve the KPs.

In summary, Model (CG) is solved by considering, successively all possible subsets *S*, and solving each S the Knapsack Problem (KPs). If number of platforms in *S* small, this procedure works fast. If it is large, the procedure is time consuming. They present an advanced algorithm that excludes a large amount of subsets *S* from consideration.

First, they introduce a concept of lex-superset. A subset S_2 is called a lex-superset of a platform subset S_1, if $S_1 \subset S_2$ with $S_1 \neq S_2$ and the platform labels of $S_2 \backslash S_1$ are larger than the largest platform label in S_1; S_2 is generated after S_1, by adding one or more platforms to S_1 with lower dual values. For example, $\{P_1, P_2, P_3\}$ is a lex-superset of $\{P_1\}$, but not of $\{P_2\}$.

Then the following steps are taken to generate "clever" subsets of platforms.

1) All platforms are sorted according to non-increasing y_i and relabeled accordingly.

2) Exceeding the range. If a platform subset S satisfies $d_{f(S)} > R$, then S and all its lex-supersets are discarded from consideration for Traveling

Salesman Tour.

3) Exceeding the capacity. If the current S is a proper subset of P, $S \neq P$, and $\sum_{i \in S} D_i \geq C$, then all lex-supersets of S are excluded form consideration for (KPs).

4) Exceeding a lower bound. To find out whether any of the lex-supersets of S will give a better solution to (CG) than the best solution found so far, a lower bound for (CG) is calculated for all lex-supersets of S.

The result obtained through the models and algorithms above is a lower bound for the original problem. In order to have a feasible solution (upper bound) to the problem, the authors propose several methods. The first one is a rounding procedure: they enforce fractional number of the variables to be one or zero in accordance with some rules to keep feasibility of the solution. The other algorithms they provide in the paper are heuristics including Cluster-and-Route algorithm and Free-Tree Heuristics. Computational experiments are based on these algorithms with the sweep algorithm and Clark-Wright algorithm as well. The results show that no algorithm outperforms others in all instances of the problem.

2.2 Review on the Column Generation technique

2.2.1 Outline of the Column Generation technique

Since Ford and Fulkerson [16] first suggested deal only implicitly with the variables of a multi-commodity flow problem over four decades ago, great progress

has been made in this research field. Dantzig and Wolfe [17] utilized this fundamental idea to develop a strategy to extend a linear program column-wise as needed in the solution process. It is Gilmore and Gomory first to put this technique to actual use as part of an efficient heuristic algorithm for solving the cutting stock problem in 1960's [18, 19]. Nowadays, column generation is becoming a prominent method to cope with problems with a huge number of variables. Furthermore, in order to obtain the integer feasible (optimal) solution, Desrosiers, Sourmis and Desrochers design an approach to embed column generation techniques within a linear programming based branch-and-bound framework [20]. They use this method to solve a vehicle routing problem with time windows for the first time.

Besides the milestone-like work mentioned above, numerous integer programming column generation applications are also described in the literature, as shown in Table 2.1 [21]. In this review on the column generation technique, we focus on not only its algorithmic side but also the application side, which are mainly the applications of column generation in some routing problems.

Table 2.1: Some application of integer programming column generation

Reference(s)	Application(s)
Agarwal et al. (1989); Desaulnier et al.(2001); Desrochers et al. (1992); Lobel (1997 1998); Riberio and Soumis (1994).	various vehicle routing problems
Borndorfer and lobel (2001); Desaulnier et al.; Desrochers and Soumis (1989).	crew scheduling;
Desrosiers et al. (1984)	multiple traveling salesman problem with time windows
Krumke et al. (2002)	real-time dispatching of automobile service units
Lubbecke (2001); Lubbecke and Zimmermann (2003); Sol (1994)	multiple pickup and delivery problem with time windows
Anbil et al. (1998); Crainic and Rousseau(1987); Vance et al. (1997)	airline crew pairing
Barnhart and Schneur (1996)	air network design for express shipment service
Erdmann et al. (2001)	airline schedule generation
Barnhart et al. (1998); Desaulnier et al. (1997)	fleet assignment and aircraft routing and scheduling
Crama and Oerlemans(1994)	job grouping for flexible manufacturing systems
Eben-Chaime et al. (1996)	grouping and packaging of electronic circuits
Park et al. (1996)	bandwidth packing in the telecommunication networks
Ribeiro et al. (1998)	traffic assignment in satellite communication systems
Sankaran (1995)	course registration at a business school
Vanderbeck(1994)	graph partitioning e.g., in VLSI, compiler design
Vanderbeck(1994)	single-machine multi-item lot-sizing
Hurkens et al.(1997); Vance (1998); Vanderbeck (1999)	bin-pack and cutting stock problems
Alvelos and Carvalho (2000)	integer multi-commodity flows
Bourjolly et al.(1997)	maximum stable set problems
Hansen et al. (1998)	probabilistic maximum satisfiability problem
Johnson et al. (1993)	minimum cut clustering
Mehrotra and Trick (1996)	graph coloring
Savelsbergh (1997)	generalized assignment problem

Given a linear program as follows which we call the master problem (MP):

$$\begin{aligned} \min \quad & z = \sum_{j \in J} c_j \lambda_j \\ & \text{s.t.} \sum_{j \in J} \mathbf{a}_j \lambda_j \geq \mathbf{b} \\ & \lambda_j \geq 0, \quad j \in J \end{aligned} \tag{2-16}$$

When using simplex method to obtain the optimal solution to the problem iteratively, we look for a non-basic variable to price out and enter the basis. In other words, given the non-negative vector **u** of dual variables, we try to find

$$\arg\min\{\vec{c}_j = c_j - \mathbf{u}^T \mathbf{a}_j \mid j \in J\}. \tag{2-17}$$

Since the complexity of this pricing step is $O(|J|)$, it is costly when $|J|$ is large. In other scenarios, sometimes we cannot express the set J explicitly. Therefore, we resort a reasonably small subset $J' \subseteq J$ of columns, resulting in the customary notion of restricted master problem (RMP). Let $\bar{\boldsymbol{\lambda}}$ and $\bar{\mathrm{u}}$ be the primal and dual optimal solutions of RMP respectively. We use the following sub-problem to generate the new columns to enter the basis and the respective cost coefficient c_j as well.

$$\bar{c}^* = \min\{c(\mathbf{a}) - \bar{\mathbf{u}}^T \mathbf{a} \mid \mathbf{a} \in \mathbf{A}\} \tag{2-18}$$

Where $\mathbf{a}_j, j \in J$ are elements of a set A. This sub-problem is feasible, for otherwise the master problem would be empty as well. If the solution to the sub-problem is non-negative, which means no reduced cost coefficient $\bar{c}_j$ is greater than or equal to zero, $\bar{\boldsymbol{\lambda}}$ optimally solves the master problem. Otherwise, we extend the RMP by a column derived from the optimal solution to the sub-problem, and repeat to

re-optimize the restricted master problem. For its role in the algorithm, (2-18) is also called the generation problem, or the column generator.

2.2.2 Formulations of the Master Problem

In applications, constraint matrices of linear programming have some features like sparse or structure in the form of large sub-matrices of zeros. This is due to the fact that activities associated with variables connect directly to only a few of conditions represented by the constraints. Hierarchical, geographical or logical segmentation of a problem can be reflected in the formulation. Therefore, we group non-zeros in such a way that independent subsystems of variables and constraints appear, possibly linked by a distinct set of constraints and/or variables. Such properties are often seen in the multi-commodity flow formulations for vehicle routing and crew scheduling problems.

The function of decomposing the original problem is to treat the linking structure at a superior, coordinating, level and to independently address the subsystem(s) at a subordinated level, exploiting any special structure at the algorithm level. In order to take advantage of the structure of the problems, it is common to combine column generation with the well-known Dantzig-Wolfe decomposition to solve the problem efficiently.

We briefly refresh the classical decomposition principle in linear programming, which is developed by Dantzig and Wolfe. It has become part of the mathematical

programming standard library. Let us consider a linear program:

$$\begin{aligned} \min \quad & z = \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathrm{D}\mathbf{x} \geq \mathbf{d} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{2-19}$$

which is named the original or compact formulation. Let $P = \{\mathbf{x} \in \mathbf{R}^n \mid \mathbf{D}\mathbf{x} \geq \mathbf{d}, \mathbf{x} \neq \mathbf{0}\} \neq \phi$. Minkowski and Weyl Theorems enable us to represent each $\mathbf{x} \in \mathbf{P}$ as convex combination of extreme points $\{\mathbf{P}_q\}_{q \in Q}$ plus non-negative combination of extreme rays $\{\mathbf{P}_r\}_{r \in R}$ of *P*, e. g.,

$$\mathbf{x} = \sum_{q \in Q} \mathbf{p}_q \lambda_q + \sum_{r \in R} \mathbf{p}_r \lambda_r, \quad \sum_{q \in Q} \lambda_q = 1, \ \boldsymbol{\lambda} \in \mathbf{R}_+^{|Q|+|R|} \tag{2-20}$$

where the index sets *Q* and *R* are finite. Replacing x in (2-19) and applying the linear transformations $c_j = \mathbf{c}^T \mathbf{p}_j$ and $\mathrm{a}_j = A\mathbf{p}_j, \ j \in Q \cup R$ we obtain an equivalent extensive formulation

$$\begin{aligned} \min \quad & z = \sum_{q \in Q} c_q \lambda_q + \sum_{r \in R} c_r \lambda_r \\ \text{s.t.} \quad & \sum_{q \in Q} a_q \lambda_q + \sum_{r \in R} a_r \lambda_r \geq \mathbf{b} \\ & \sum_{q \in Q} \lambda_q = 1 \\ & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned} \tag{2-21}$$

It typically has a tremendous number |Q|+|R| of variables, but possibly substantially fewer rows than (2-19). The equation $\sum_{q \in Q} \lambda_q = 1$ is the convexity constraint. If $\mathbf{x} \equiv 0$ is feasible for P in (2-21) without any cost, it may be omitted in *Q* and hence the convexity constraint becomes $\sum_{q \in Q} \lambda_q \leq 1$ in the model. One thing should be noticed here is that although the compact and the extensive formulations are equivalent in that they have the same optimal objective function value z, the

respective polyhedra are not combinatorially equivalent. Since in (2-20), **x** is uniquely represented by a given $\boldsymbol{\lambda}$, but not vice versa.

So far we reformulate the model (2-19) as (2-21), which is a special master problem. In (2-21), the objective function is linear, and the set of columns is implicitly defined by the extreme points and extreme rays of a convex polyhedron *P*. It is efficient to utilize column generation to solve this problem. The corresponding RMP with current subsets $Q' \subseteq Q,\ R' \subseteq R$ in (2-21) has a dual optimal solution $\overline{\mathbf{u}}, \overline{v}$, where variable *v* corresponds to the convexity constraint. The pricing problem in Dantzig-Wolfe decomposition now is to determine

$$\overline{c}^* = \min\{(c^T - \overline{\mathbf{u}}^T A)\mathbf{x} - \overline{v} \mid D\mathbf{x} \geq \mathbf{d}, \mathbf{x} \geq \mathbf{0}\} \qquad (2\text{-}22)$$

(2-22) is a linear program again. When $\overline{c}^* \geq 0$, no negative reduced cost column can be found, and the algorithm terminates. When $\overline{c}^* < 0$, the optimal solution to (2-22) is an extreme point Pq of P, and we add the column $[\mathbf{c}^T\mathbf{p}_q, (A\mathbf{p}_q)^T, 1]^T$ to the RMP. When $\overline{c}^* = -\infty$, an extreme ray Pr of P as a homogeneous solution to (2-21) and we add the new column $[\mathbf{c}^T\mathbf{p}_q, (A\mathbf{p}_q)^T, 0]^T$ to the RMP. Note that the algorithm is finite as long as finiteness is ensured in optimizing the RMP. Dantzig-Wolfe type approximation algorithms with guaranteed convergence rates have been proposed for certain linear programs, readers can see the reference given therein.

One application of combining Dantzig-Wolfe decomposition with column generation in the literature is proposed by Savelsbergh for solving the generalized assignment problem (GAP) [22].

Another common formulation of Master Problem is based on set-partitioning. This is due to the properties of one-one mapping relationship between different items. This kind of model is easy to form since it reflects the relationship between variables naturally and often can be seen in various vehicle routing problems. We will introduce two applications of set-partitioning based column generation method in different domains.

First, we will give a combinatorial description of set partitioning problem. Let M be a non-empty and finite set. Let F be a family of acceptable or feasible subsets of M. Associated with each family j of F is a cost c_j. The problem is to find a collection of members of F, which is a partition of M, where the cost sum of these members is minimal.

An integer programming formulation of the set-partitioning problem reads

$$\begin{aligned} \text{(SPP)} \quad & \min \ z = \mathbf{c}^T \mathbf{x} \\ & \text{s.t.} \ \ \mathbf{A}\mathbf{x} = \mathbf{1} \\ & \qquad \mathbf{x} \le \mathbf{1} \end{aligned} \tag{2-23}$$

Where $\mathbf{x}$ is a solution vector, $0 \le c \in \mathbf{R}^n$ a cost vector, and $A \in [0,1]^{m \times n}$ a zero-one matrix. M corresponds to the m rows of matrix A and the subsets of M correspond to the columns of this matrix in such a way that $a_{ij} = 1$ if $i \in j$ and $a_{ij} = 0$ if $i \notin j$. The stipulation that each member of M has to be covered once corresponds to the constraint set of (1.1), which defines F. The SPP is a well-known NP-hard problem.

Now we discuss the first one application of column generation based on set partitioning, which is presented by Lorena and Senne. They use this approach to solve the Capacitated p-Median Problems (CPMP). The Capacitated p-Median Problem refers to a set $I = \{1,...,n\}$ of potential locations for p facilities, a set $J=\{1,..., m\}$ of customers, and nX m matrix (g_{ij}) of transportations costs for satisfying the demands of the customers from the facilities. The capacity of each possible median is Q. The capacitated p-median problem is to locate the p facilities at locations of I in order to minimize the total transportation cost for satisfying the demand of the customers. Each customer is supplied from the closest open facility. Lorena and Senne apply the column generation method to the problem.

The master problem is thus rewritten as:

$$\begin{aligned} \text{(CPMP)} \quad & \min z = \sum_{k=1}^{m} c_k m_k \\ & \text{s.t.} \ \sum_{k=1}^{m} A_k x_k = 1 \\ & \sum_{k=1}^{m} x_k = p \\ & x_k \in [0,1], \ k = 1,...,m. \end{aligned} \tag{2-24}$$

Where

$S = \{S_1, S_2, ..., S_m\}$, is a set of subsets of N;

$A = [a_{ik}]_{m \times n}$, is a matrix with $a_{ik} = \begin{cases} 1, & \text{if } i \in S_k \\ 0, & \text{otherwise} \end{cases}$, satisfies $\sum_{i \in N} q_i a_{ik} \le Q$;

and $c_k = \min_{i \in S_k^1} (\sum_{j \in S_k^1} d_{ij})$, considering $S_k^1 = \{i \in S_k \mid a_{ik} = 1\}$

The other set-partitioning based master problem of column generation we will introduce next is a formulation for the Vehicle Routing Problem proposed by Agarwal

et al. [23].

The master problem they provide is:

$$\begin{aligned} \text{(SP1)} \quad & \min \ z = \sum_{j \in \Omega} c_j x_j \\ & \text{s.t.} \ \sum_{j \in \Omega} a_j x_j = e \\ & x_i \in \{0,1\} \end{aligned} \tag{2-25}$$

In this SP formulation, Ω is the possible feasible route set, and each vehicle route is represented by a binary n-vector a_j. The element a_{ij} of vector a_j is 1 if demand point I is visited on route a_j, otherwise 0. A cost c_j represents the total distance traveled on the route a_j.

2.2.3 Discussion on the pricing problem

One difficulty in the column generation lies in how the sub-problem is formed to search virtually all non-basic columns. In fact, those vectors $\mathbf{a} \in \mathbf{A}$ in master problem usually represent combinatorial objects like paths, feasible crew schedules or sets. Therefore, we can define **A** and the interpretation of cost on these structures and have a valuable information about what the appearance of the possible columns are. Taking the classic stock cutting problem for instance, one-dimensional cutting stock problem is defined by the following data: $(m, L, l = (l_1, ..., l_m), b = (b_1, ..., b_m))$, where L denotes the length of each stock piece, m denotes the number of smaller piece types and each type i=1,…,m, l_i is the piece length, and b_i is the order demand. In a cutting plan we must obtain the required set of pieces from the available stock length. The objective is to minimize the number of used stock length. Gilmore and

Gomory develop a mathematical model utilizing the column generation to solve this problem for the first time [18, 19]. The formulation is as follows:

$$\begin{aligned} \text{RMP} \quad & \min \sum_{j \in J} Y_j \\ & \text{s.t.} \sum_{j \in J} a_{ij} Y_j \geq b_i \quad i = 1,...,m; \\ & Y_j > 0, \quad j \in J \\ & Y_j : \text{number of times pattern } j \text{ is used;} \\ & a_{ij} : \text{number of times item } i \text{ is cut in pattern } j; \\ & J : \text{set of cutting pattern.} \end{aligned} \tag{2-26}$$

The pricing problem is:

$$\begin{aligned} \max \quad & \sum_{i \in I} \pi_i a_i \\ & \text{s.t.} \sum_{i \in I} l_i a_i \leq L; \\ & a_i \geq 0, \text{integer}, i \in I. \\ & \pi_i : \text{dual variable from the RMP.} \end{aligned} \tag{2-27}$$

In one-dimension cutting stock problem, the sub-problem is a typical Knapsack Problem, which generates new columns to enter the restricted master problem iteratively until its objective function value is less than or equal to zero (since it is a maximum problem).

The role of the pricing problem is to provide a column that prices out profitably or prove that no such column exists. It is important to note that any column with negative reduced cost helps achieve this aim. Especially, we do not need to solve the sub-problem (2-17) exactly, an approximation is sufficient until the last iteration. We may add many negative reduced cost columns from a sub-problem, and sometimes even positive ones are used. Desrochers et al. solve a temporary of the sub-problem,

or relaxation when they cope with the vehicle routing problem with time windows [24].

One important concept in column generation is dominance and redundancy of columns. A column with reduced cost $\bar{c}$ is dominated if there exists another column with reduced cost not greater than $\bar{c}$ for all dual variables ranging within their respective domains. A column a_s is called redundant if the corresponding constraint is redundant for the dual problem. That is,

$$\mathbf{a}_s = \sum_{r \subset s} \mathbf{a}_r c_r \text{ and } c_s \geq \sum_{r \subset S} c_r \lambda_r \tag{2-28}$$

Sol [25] discloses a characterization of redundant columns in the case of identical sub-problems and a proof that there is no redundant column if all sub-problems are distinct. For set partitioning problems with identical sub-problems, we can use an alternative pricing rule to avoid generating the redundant columns. These rules include Steepest-edge pricing, the practical Devex variant and the lambda pricing rule [26, 27, 28].

Pricing rules are sensitive to the dual variable values when there exist non-unique dual solutions. For large set partitioning problems, which are usually highly primal degenerate, the value of dual variables are not so efficient in producing new column to adjoin to the RMP. Therefore, the key issue for that kind of problem is to overcome the degeneracy.

2.2.4 Tailing-Off effect

One of the drawbacks of the column generation technique is its poor convergence, especially in some degenerated problems, i.e., the Vehicle Routing Problem. While sometimes a near optimal solution is approached very quickly, in general only little progress can be obtained per iteration. Graphically speaking, the solution process exhibits a long tail (Gilmore and Gomory) [18] before the optimal solution is obtained. This phenomenon is called the *tailing off effect.* Several approaches called column generation stabilization are proposed in the literature to overcome this inherent drawback of column generation technique. Agarwal et al. [23] present a simple idea to specify bounds of the dual variable values by using a heuristic solution to the VRP, such as the one obtained by the Clarke and Wright algorithm [29]. A statistical model is proposed to estimate good starting values for the dual variables. Marstern et al. [30] introduce a Boxstep method to have a more sophisticated control of the dual variables. The principle of their method can be stated as follows: let $\bar{u}$ represent an optimal solution to the current restricted dual Restricted Master Problem (RMP). Dual variables can be constrained to stay in a "box around $\bar{u}$" if lower and upper bounds are imposed respectively. Then, the *RMP* is re-optimized. If the new dual optimum is attained on the boundary of the box, we have a direction where a box should be relocated. Otherwise, the optimum is obtained in the interior of the box, which produces the sought global optimum. du

Merle et al. [31] provide a stabilization approach that includes a more flexible, linear programming concept based box, together with an ε-perturbation of the right hand side of the constraints. All numerical results of these methods show that the stabilized approaches can be used to improve the solution time.

2.2.5 Integer Programs and column generation

Column generation technique has been successful in solving large-scale linear programming. For mixed integer programs, a good method requires formulations whose linear programming relaxation gives a good approximation to the convex hull of feasible solutions. In the past twenty years, the “branch-and-cut” method has been paid a great deal attention to and quite a few outcomes have been achieved (Hoffman and Padberg, Nemhauser and Wolsey) [32, 33].

The idea behind the branch-and-cut is as follows. In order to handle the LP relaxation of the original MIP efficiently, we leave out some classes of valid inequalities from the problem because it has too many constraints. This will yield infeasible solutions to the problems. Therefore, a sub-problem called the separation problem is solved to try to identify violated inequalities in a class. If violated inequalities are found, some of them are added to the LP to cut off the infeasible solution. Then the LP is re-optimized. If we cannot find violated inequalities, branching is performed. Branch-and-cut is a generalization of branch-and-bound with LP relaxations and allows separation and cutting to be applied throughout the

branch-and-bound tree.

In the last decade, a new method to the MIP called “branch-and-price” is presented by Desroscher et al. when they solve the vehicle routing problem with time windows for the first time. The principle of branch-and-price is similar to that of branch-and-cut except that the procedure focuses on column generation rather than row generation. Actually, these two are complementary procedures for tightening an LP relaxation.

The branch-and-bound algorithm is based on the column generation technique. When column generation procedure cannot find negative reduced cost, the LP relaxation obtains its optimal solution. Branching occurs when the LP solution does not satisfy the integrality conditions. Like branch-and-cut, branch-and-price is also a generalization of branch-and-bound with LP relaxation, allowing column generation applied all through the branch-and-bound tree.

Some important issues need to be concerned in branch-and-price method include lower bound and early termination, and the branching strategy. In each node of a branch-and-bound tree, we derive lower bounds on the best possible integer solution in the respective branch from solving the RMP linear relaxation by column generation. It is naturally to expect that the tailing off effect should be amplified when the size of linear programs is very large. We have a simple amendment for the need of integer solutions: Stop generating columns when tailing off effect happens and perform a branch decision. This early terminating is based on the following.

Assuming $c_j \in Z, j \in J$, which for rational data is no loss of generality, column generation can be stopped as soon as $\lceil LB \rceil = \lceil \bar{z} \rceil$. Due to this purpose they have been widely used in the literature, i.e., Sol [34]; Vanderbeck [35]; Vanderbeck and Wolsey [36].

Early termination makes the algorithm effective for integer programs in contrast to linear programs. We can even terminate heuristically early than $\lceil LB \rceil = \lceil \bar{z} \rceil$. Therefore, a tradeoff should be considered between computational efforts and the quality of the obtained lower bound upon premature termination.

As to the branching strategy, a valid branch scheme divides the solution space in such a way that the current fractional solution is excluded, integer solutions stay intact, and finiteness of the algorithm is ensured. Furthermore, some general rules of thumb prove useful, i.e. producing branch of possibly equal size, which is referred to as balancing the search tree. Important decision should be made early in the tree. In particular, when the master problem has to be solved integrally, a compatible branching scheme is sought which prevents columns that have been branched on from being regenerated without a significant complication of the pricing problem [37]. This would generally lead to finding the *k*th best sub-problem solution instead of the optimal one [38].

As to general branching scheme in case that the master problem has to be solved integrally, Barnhart et al. and Vanderbeck have made important work. The most common strategy in conjunction with column generation is Ryan and Foster's

designed for set partitioning problems, which is included in the following proposition.

Proposition 2.8 Given $A \in \{0,1\}^{m \times |J^1|}$ and a fractional basic solution to $A\boldsymbol{\lambda} = \mathbf{1}, \boldsymbol{\lambda} \geq \mathbf{0}$. Then there exist, $r, s \in \{1,...,m\}$ such that $0 < \sum_{j \in J^1} a_{rj} a_{sj} \lambda_j < 1$.

This proposition shows that when such two rows are identified, we obtain one branch in which these rows must be covered by the same column, e.g., $\sum_{j \in J^1} a_{rj} a_{sj} \lambda_j = 1,$ and one branch where they must be covered by two distinct columns, e.g., $\sum_{j \in J^1} a_{rj} a_{sj} \lambda_j = 0.$ This information can be transferred to and obeyed by the pricing problem without any difficulty.

Besides the pioneer work above, Barnhart et al. and Vanderbeck **[38, 39]** present the principles and guidelines of the branch and price approach in different scenarios. The principle for the branch and price approach can be summarized as follows: first, if necessary, use Dantzig-Wolfe decomposition to rewrite the original formulation in accordance with the property of the problem into two sub-problems, namely, the master problem and the pricing problem. Next, the column generation approach is performed to obtain the optimal solution to the LP relaxed master problem. Then, different branching schemes may be adopted and carried out to find the integer solutions. Generally speaking, for integer column generation method, three branching schemes are given in [39]. Rule A is: enforce $\sum_{q \in Q(k): q_i^k \geq v} \lambda_q \in \{0,1\} \ \forall k, i, v.$ λ_q is the combinatorial coefficient in Dantzig-Wolfe decomposition. $Q(k)$ is the integer polyhedron of the kth supproblem, i is the index of the strip width, and v is the

number of strips of the width w_i. Rule B is: enforce $\sum_{q\in Q:q_i\geq v}\lambda_q$ integer $\forall i,v.$, while rule C is: enforce $\sum_{q\in Q:q\leftrightarrow q'\&q_l'=1}\lambda_q$ integer $\forall l\in\{1,...,n'\}$. Rule C can be illustrated as follows [39]: given a fractional solution λ, we search for a index l, i.e., a component of the 0-1 form of columns q, such that the number of columns with entry one in that component, $\sum_{q\in Q:q\leftrightarrow q'\&q_l'}\lambda_q$ is fractional and thus is enforced to be integer. For the general assignment problem, the branch scheme is [38]: enforce $\sum_{k:y_{rk}=1,y_{sk}=1}\lambda_k\in\{0,1\}$. r and s are the row numbers in the master problem, k is the column number in the master problem.

This pioneer work also generates a powerful insight which is used already in standard branch-and-bound, that is, to branch on meaningful variable sets. The most valuable source of information is those original variables of the compact formulation. They must be integer, and they are what we branch and cut on. Branching and cutting decisions both involve the addition of constraints. We may require integrality of x at any node of a branch-and-bound tree, but it is not efficient. Hurkens et al. propose a problem specific penalty function method [40]. Alternatively, given an added set $H\mathbf{x}\geq h$ of constraints, these restrictions on the compact formulation can be incorporated in $A\mathbf{x}\geq\mathbf{b}$, in $\mathbf{x}\in\mathbf{X}$, or partially in both structures. In any scenario, the new problem is of the general form of the compact formulation. The new RMP is still a linear program, and the earlier the sub-problem structure is tractable, the less severe complication we will face.

It is important to be aware that even if a new decision set goes into the master

problem structure, the pricing problem may change. Some examples are given in the routing and scheduling area. Ioachim et al. have found that linear combinations of time variables appear in the master problem structure which results in the consequence that these time variables also appear in the objective function of the sub-problem together with the flow variables. This changes the way to solve the constrained shortest path problem.

Another issue is the implementation of a column generation based integer programming code. All strategies from standard branch-and-bound apply, including depth first search for early integer solutions, heuristic fathoming of nodes, rounding and fixing of variables, and many more [41]. New columns are generated at any node of the tree.

Concluding, no efficient way of handling the difficulty of finding an optimal integer solution to a problem solved using a column generation scheme is available two decades ago. Today, it is no longer true when we obtain the compact formulation of the problem and generate columns at each node the search tree. This fundamental and simple approach has been in use for nearly twenty years and is being refined ever since. The price we have to pay for this simplicity is that besides RMP, sub-problem, and branch-and-bound also the compact formulation has to be represented in order to recover a solution in terms of the original variable **x**.

CHAPTER III

A TWO-STAGE EXACT ALGORITHM TO THE SPLIT DELIVERY VEHICLE ROUTING PROBLEM WITH VALID INEQUALITIES

In this chapter, we present an exact approach to the Split Delivery Vehicle Routing Problem when the number of the vehicle in the fleet is fixed, namely, the smallest one that satisfies the total demand. In Section 3.1, we provide the formulation of the new method. In Section 3.2, a class of efficient valid inequalities for the model and the complete algorithm are illustrated; we display the computational experiment results in Section 3.3. Finally, discussion on the possible future work in this problem is included in Section 3.4.

3.1 A Two-Stage Formulation for the SDVRP

Like the CVRP, the SDVRP is also an NP-hard problem [1]. Thus, most work in this field handles with some simplified sub problems rather than the whole original formulation. Dror et al.'s [8, 9] cutting-plane algorithm can be used to solve a relaxation of the SDVRP without considering the sub-tour elimination constraints firstly. Then they include routes that have sub-tours in them and other efficient

inequalities in the first sub problem to solve it again. These steps repeat until no violated constraints are identified. Finally, a branch-and-bound algorithm is applied to the problem to obtain the integer solution. Belenguer et al. [10] first deal with a reduced SDVRP (R_{SDVRP}) that ignores the index of the vehicle and the sub-tour elimination constraints as well, and used several heuristics algorithms to identify the violated constraints. In this paper, we will develop valid inequality to finally solve the problem of the index of the vehicles.

To reduce the size of models, we propose a two-stage algorithm for the SDVRP in this paper. We assume the number of the fleet of the vehicles is fixed, equaling to the minimum required number of the vehicles to fulfill all demands, and the demand at each point is allowed to be larger than the capacity Q of a vehicle.

The first stage model C1 is a clustering problem to assign the demands to vehicles without considering travel distance costs.

$$C1:\ \min \sum_{k=1}^{U} V_k$$

$$s.t.\ \ w_{ik} \le a_i y_{ik}, \quad i = 1,...,N, k = 1,...,U \tag{3-1}$$

$$\sum_{k=1}^{U} w_{ik} = a_i, \quad i = 1,...,N \tag{3-2}$$

$$\sum_{i=1}^{N} w_{ik} \le 1, \quad k = 1,...,U \tag{3-3}$$

$$y_{ik} : binary; w_{ik}, V_k \ge 0$$

From now, we normalize the demand and capacity without loss of the generality, here $a_i = d_i/Q$.

$$y_{ik} = \begin{cases} 1, & \text{if supplier } i \text{ is visited by truck } k; \\ 0, & \text{otherwise;} \end{cases}$$

w_{ik} = normalized load picked up at supplier i by vehicle k.

V_k: distance lower bound of vehicle k.

U: the minimum number of the vehicles that satisfies the total demand of the points.

N: number of the points.

C1 is an assignment problem, and it yields a feasible clustering solution meeting all demands under capacity constraints. Without any constraints on V_k, they are all zeros in the first iteration and their sum provides a lower bound for the overall SDVRP. The second stage problem T is a typical Traveling Salesman Problem (TSP) for each vehicle and provides the cost for each cluster. Assume K_l is the set of the routes used in the solution to the first stage at the lth iteration and y_{ik}^l is the solution to the first stage at the lth iteration. We define

$$I_k^l = \{i \mid y_{ik}^l = 1\} \text{ for vehicle } k \in K_l \text{ at iteration } l$$

For each I_k^l obtained from C1, sovle the *TSP* below :

$$\text{T}: \ \min z_k^l = \sum_{i \in I_k} \sum_{j \in I_k} c_{ij} x_{ij}^k$$

$$s.t. \ \sum_{i \in I_k^l} x_{ij}^k = 1, \ \ \forall \ j \in I_k^l \qquad (3\text{-}4)$$

$$\sum_{j \in I_k} x_{ij}^k = 1 \quad \forall \ i \in I_k^l \qquad (3\text{-}5)$$

$$\beta_i^k - \beta_j^k + N_k x_{ij}^k \le \left| I_k^l \right| - 1 \ (\text{for } i \ne j; i \in I_k^l \setminus \{0\}; j \in I_k^l \setminus \{0\}) \qquad (3\text{-}6)$$

$$x_{ij}^k = 0 \text{ or } 1, \ u_j^k \ge 0$$

where

$\left|I_k^l\right|$: the number of suppliers served by vehicle k at iteration l;

$x_{ij}^k : \begin{cases} 1: \text{if truck } k \text{ visits demand point } j \text{ just after demand point } i \\ 0: \text{otherwise} \end{cases}$;

β_i^k : variables to prevent subtour.

z_k^l is the travel distance cost for cluster I_k^l and $\sum_{k \in K_l} z_k^l$ yields an upper bound for the SDVRP. Although the TSP itself is an NP-hard problem, model T is typically a small problem in practice, and we can use commercial optimization software like CPLEX 9.0 to obtain the solution very quickly (much less than 1 second). Therefore, this paper will focus on the interaction between the two stages and the more efficient way to solve the first stage model.

Unlike other clustering-first, routing-second constructive heuristics, this algorithm considers the feedbacks from the second stage and adds them as new constraints (cuts) to the first stage if there are new clusters. A new lower bound can be obtained by solving the first stage problems with the added cuts. For each set I_k^l, we create the following cuts:

$$\sum_{i \in I_w^l} y_{ik} \le \rho_{wk}^l + \left|I_w^l\right| - 1 \qquad w = 1,...,U,\ k = 1,...,U \tag{3-7}$$

$$z_w^l \rho_{wk}^l \le V_k \qquad w = 1,...,U;\ k = 1,2,...,U \tag{3-8}$$

When vehicle w visits all demand points in set I_k^l, then the total travel distance of the vehicle should exceed z_k^l. Although ρ_{wk}^l, indicating whether vehicle w visits all demand points in set I_k^l, should be a binary variable physically, we can relax it to

a continuous variable because y_{ik} is a binary variable and $\left|I_w^l\right|-1$ is an integer. In fact, we can even combine (3-7) and (3-8) into a single constraint:

$$V_k \geq z_w^l(\sum_{i\in I_w^l} y_{ik} - \left|I_w^l\right| + 1) \qquad w = 1,2,...,U; k = 1,2,...,U \tag{3-9}$$

Because $\sum_{i\in I_w^l} y_{ik} - \left|I_w^l\right| + 1 \leq 1$, and V_k are defined as nonnegative variables.

Since the cuts are characterized by the set I_k^l, the set of all cuts at current iteration *h* is defined as $\Omega = \{I_k^l \left| k \in K_l, l = 1..h\right.\}$. With added constraints (3-9), the first stage model C1 is solved again to get a new lower bound. In each iteration, the lower bound always decreases or keeps the same because of more constraints. To avoid some computational repetition, we redefine K_l as the set of routes that are used in the solution to problem C1 in the current iteration, but are not included in Ω.

$$K_l = \{I_k^l \left| I_k^l \notin \Omega\right.\}$$

We implement the algorithm with CPLEX 9.0 and find the convergence rate is low with the algorithm. In the early iterations, some demand points that are far away from each other are grouped in the same cluster. To reduce the number of iterations, triangular inequalities mentioned above are introduced in the first stage problem C1 in the first iteration.

$$V_k \geq C_{0i}y_{ik} + C_{0j}y_{jk} + C_{ij}(y_{ik} + y_{jk} - 1), \ \forall i, \forall j, \forall k; \tag{3-10}$$

When the problem size increases, the number of triangular inequalities significantly increases because the number of the inequalities is *N(N-1)/2*. In fact, triangular inequalities are only introduced to avoid the clusters with the suppliers far

away from each other.

Therefore, instead of using all the triangular inequalities, we rank the perimeter of these triangles in a descending order and only select the first half of these inequalities. Numerical experiment shows a significant improvement of the speed of the algorithm.

3.2 Valid inequalities for Two-Stage algorithm to the SDVRP

We use commercial optimization software CPLEX9.0 to solve both stage models. CPLEX basically uses branch-and-bound to solve integer program models with some general fractional cuts. We observe many node explorations for the first stage model in each round. The lower bound provided by linear relaxation is so loose that numerous branches are required. Therefore, in addition to the triangular inequalities, the following classes of constraints are also valid for the first stage model. We find constraint (3-1) $w_{ik} \le a_i y_{ik}, \forall i, \forall k$ can yield small y_{ik} in the SDVRP (though it is not a problem for the CVRP) when we relax the integer requirement on y_{ik} and splits occur. For example, if both vehicle 1 and 2 visit demand 1 and each picks up one half of the demand, both y_{11} and y_{12} will be 0.5. Then the related triangular cuts and the cuts obtained from the second stage will not work under linear relaxation because of too loose lower bound for V_k. The problem will become worse, when the demand at one point is larger than the capacity of a vehicle, because w_{ik} will be much smaller than a_i. In fact, no matter how large the demand at

point i is, $\sum_{k=1}^{v} y_{ik}$ will always be 1 in the linear relaxed model. Based on this observation, we develop the following two valid inequalities.

1) Required Number of Vehicles Valid Inequality for points with large demand

According to the definition of the SDVRP, each demand point should be satisfied. Therefore, it is valid to include the following inequalities in the model:

$$\sum_{k=1}^{v} y_{ik} \geq \lceil a_i \rceil \quad for\ i = 1,...,N \tag{3-11}$$

When the demand of a point is larger than the capacity Q, this inequality can improve the lower bound. For example when $a_i = 1.2$, then a valid inequality $\sum_{k=1}^{v} y_{ik} \geq 2$ can be added into the first stage.

2) Non-idleness of the vehicle inequality

Since each vehicle must visit at least one demand point, the following inequalities are valid:

$$\sum_{i=1}^{N} y_{ik} \geq 1 \quad \text{for } k = 1,...,U \tag{3-12}$$

In the next part, we will develop some more powerful valid inequalities. The following inequality derives from Theorem 2.1.

3) Optimal Solution property inequality:

According to Theorem 2.1, inequalities (3-13) are valid:

$$y_{iw} + y_{iv} + y_{jw} + y_{jv} \le 3, \quad i, j = 1,..., N; \ w, v = 1,..., U, w \ne v, i \ne j \tag{3-13}$$

4) Vehicle index assignment valid equality/inequality

When all vehicles are identical in the fleet, an SDVRP model has numerous equivalent solutions with the same routes and pickup but different vehicle indexes. Dror and Trudeau **[1]** give a valid inequality of $\sum_{j=1}^{N} x_{1j}^{1} = 1$ to make sure the first vehicle visits the first demand point. We can have equivalent valid inequality $y_{11} = 1$ to let the first vehicle cover the first demand point. The first possible extension could be

$$\sum_{i=1}^{k} y_{ik} \ge 1 \quad k = 2,..., \min(U,N) \quad \text{with} \quad y_{11} = 1 \tag{3-14}$$

Intuitively, demand point 2 could be visited by the first vehicle, which visits demand point 1 or not. If not, we can assign demand point 2 to vehicle 2. Though (3-14) is a pretty strong valid inequality, we can even develop a stronger one. If it is assumed that $a_1=0.6$ and $a_2=0.6$, more than one vehicle are required to visit demand point 1 and demand point 2. We can set $y_{11}=1$ and also set $y_{22}=1$.

Lemma 3.1: If $a_1 + a_2 > 1$, $y_{11}=1$ and $y_{22}=1$ are two valid equalities.

Proof: For a feasible integer solution, at least two vehicles visit demand point 1 and 2 when $a_1 + a_2 > 1$. There are totally four cases: 1) If one vehicle just visits demand point 1, and the other one just visits demand point 2, we can assign the first vehicle visiting demand point 1 as vehicle 1 and the other vehicle as vehicle 2. 2) If one

vehicle both visits point 1 and 2, and the other vehicle just visits point 2, the first vehicle is assigned as vehicle 1 and the other vehicle is set vehicle 2. 3) If one vehicle both visits point 1 and 2 and the other vehicle just visits point 1, the first vehicle is assigned as vehicle 2 and the other vehicle is set as vehicle 1. 4) If both vehicles visit both demand points, we can arbitrarily choose one vehicle as vehicle 1 and the other as vehicle 2. We can find one of these four cases for two vehicles in any feasible integer solution. Under any cases, $y_{11}=1$ and $y_{22}=1$ are valid. □

We can further extend Lemma 3.1 to theorem 3.1 about valid inequality for the vehicle index assignment. Without loss of generality, from now on the demand points are assumed to be ranked with descending demands: $a_1 \geq a_2 \geq a_3 \ldots \geq a_n$.

Theorem 3.1: If $\sum_{i=o}^{m} a_i > m - o,\ o = 1,...,m\text{-}1$, $y_{mm}=1$ is a valid equality.

Proof: Lemma 3.1 is a special case of theorem 3.1 with $m=2$. If we assume $m=2,...,t$ is valid, now we need to prove $m=t+1$ is valid. Because $\sum_{i=o}^{t+1} a_i > t+1-o,\ o = 1,...,t$ and a_i is in a descending sequence, $\sum_{i=o}^{t} a_i > t-o,\ o = 1,...,t-1$. Furthermore, the condition holds for $m=2,...,t$, so we have $y_{ii}=1,\ i=1,...,t$. In other words, we assign one vehicle for each demand point from 1 to t. In a feasible integer solution to the SDVRP, if demand point $t+1$ is visited by another vehicle not belonging to the first t vehicles, this vehicle can be assigned as $t+1$ so $y_{t+1,t+1}=1$ is true. If demand point $t+1$ is only visited by the first t vehicles based on the previous index assignment, at least

one of the remaining vehicle visits one or more than one of the first *t* demand points, since $\sum_{i=1}^{t+1} a_i > t$. Assuming one vehicle visiting demand point *r* $(r \le t)$ doesn't belong to the first assigned *t* vehicles, we reassign the new vehicle as the *r*th vehicle. After removing demand point *r*, the condition $\sum_{i=o, i\neq r}^{t+1} a_i > t - o,\ o = 1,...,r-1$ and $\sum_{i=o}^{t+1} a_i > t + 1 - o,\ o = r+1,...,t$ still hold and the condition is equivalent as $m=t$ case, so we can have one vehicle for each demand node among *1,..., r-1* and *r+1,..., t+1* and number them as vehicle *1,..., r-1* and *r+1,..., t+1*. □

For instance, if we have an SDVRP like $(a_1=1.3, a_2=1.2, a_3=0.9, a_4=0.6, a_5=0.3, a_6=0.2)$, then $\lceil 4.5 \rceil = 5$ vehicles are required. We can have the following valid equalities, $y_{11}=1$, $y_{22}=1$, $y_{33}=1$ and $y_{44}=1$ based on theorem 3. If we have an *SDVRP* like $(a_1=1.3,\ a_2=0.65,\ a_3=0.6,\ a_4=0.5,\ a_5=a_6=0.3)$, $\lceil 3.65 \rceil = 4$ vehicles are required and we can have the following valid equalities, $y_{11}=1$, $y_{22}=1$ *and* $y_{33}=1$. Here, $y_{44}=1$ is not true because $a_2+a_3+a_4=1.75 \le 2$ though $a_3+a_4>1$. In both examples, one remaining vehicle has not been assigned. In the first example of $(a_1=1.3, a_2=1.2, a_3=0.9, a_4=0.6, a_5=0.3, a_6=0.2)$, if we consider demand points 5 and 6 as one point with $a_5'=0.5$, the fifth vehicle must visit this new combo point because condition $\sum_{i=o}^{m} a_i > m - o,\ o = 1,...,m\text{-}1$ is true now for $m=5$. In other words, truck 5 must visit one or both of points 5 and 6. Therefore, $y_{55} + y_{65} \ge 1$ is a valid

inequality. For the second example, we know $\sum_{i=o}^{m} a_i > m - o,\ o = 1,...,m\text{-}1$ is true for $m=3$, but not for $m=4$. We can combine a_4 and a_5. Since $a_4 + a_5 = 0.8$, which is larger than $a2$ and $a3$, we need to re-rank the sequence as $(a1, (a4+a5), a2, a3, a6)$. Therefore, the valid inequalities are $y_{11}=1$, $y_{42}+y_{52}\geq 1$, $y_{23}=1$ and $y_{34}=1$, because the condition in theorem 3.1 is met for $m=4$ in the new sequence. On the opposite side, splitting can also be implemented. Look at the example of $(a_1=2.5, a_2=1.6, a_3=0.9, a_4=0.6)$. 6 vehicles are required, but only four valid inequalities $(y_{ii}=1, i=1,...,4)$ can be obtained based on theorem 3.1. In fact, at least three vehicles are needed to visit demand point 1, and five vehicles are required for demand point 1 and demand point 2. If we split demand point 1 into three points with $a_{11}=1.01$, $a_{12}=1.01$ and $a_{13}=0.48$ and split demand point 2 into two points with $a_{21}=1.01$ and $a_{22}=0.59$, the new sequence will be (a_{11}, a_{12}, a_{21}, a_3, a_4, a_{22}, a_{13}). Based on theorem 3.1 and the combination, we can get valid inequalities like $y_{11}=1$, $y_{12}=1$, $y_{23}=1$, $y_{14}+y_{24}\geq 1$, $y_{35}=1$ and $y_{46}=1$, so all six vehicles are assigned with an index.

Theorem 3.2: With combining and splitting demand points, an assignment valid inequality can be created for each vehicle required in an SDVRP.

Proof: With splitting and combining, finally we can let each slot in the final sequence of demand point with the demand of $1+\varepsilon$ and totally there are U=$\left\lceil \sum_{i=1}^{n} a_i \right\rceil$ slots, if we let ε be a very small positive number. Condition in theorem 3.1 can be met for $m=U$, so all U trucks have an equality if only one demand point in the slot, or an

inequality if there are more than one demand point in the slot. □

Valid inequalities created by theorem 3.2 with combination and splitting are stronger than the ones defined by (3-14) and they conflict with each other, so only the formers are recommended in the final algorithm. By assigning each vehicle to one or more demand points, numerous duplicated combinations will be avoided, and thus the speed of the whole algorithm can be significantly improved.

4) Route distance inequalities

Considering the relationship between the distance of any route V_k (k=1,…,U) and those y_{ik} (i=1,…N; k=1,…,U), we obtain some propositions.

Proposition 3.1 The constraints

$$V_k \geq 2c_{i0}y_{ik} \quad \text{for } i = 1,...,N; k = 1,...,U \tag{3-15}$$

are valid inequalities for the first stage model of the SDVRP.

Proposition 3.1 is straightforward, since every vehicle should start from and go back to the depot, and if point i is visited by the vehicle, the distance of the segment between point i and the depot is the shortest.

Proposition 3.2 The constraints

$$V_k \geq c_{i0}y_{ik} + c_{j0}y_{jk} \quad \text{for } i, j = 1,...,N; k = 1,...,U \tag{3-16}$$

are valid inequalities for the first stage model of the SDVRP.

Proposition 3.2 can be extended for any $S \subseteq N$ with the following construction algorithm:

1. Let's assume we have already create a valid inequality for a set $S \subseteq N$:

$V_k \geq \sum_{i \in S} e_i^S y_{ik}$ for $k = 1,...,U$, where e_i^s is the first $|S|$ lowest c_{uv}, where $u, v \in S \cup \{0\}$. Define E^S as the set of *(u,v)* with c_{uv} equal to some e_i^s.

2. Create a set *S'* by adding one demand point *j* which doesn't belong to *S* into *S*. Rank c_{ij} with ascending order and let $E^{S'}$ the set of *(u,v)* with *the first |S|+1 lowest* c_{uv}, where $u, v \in S' \cup \{0\}$.

3. Let

   $e_j^{S'} = \min c_{uj}$, if $\min c_{uj} \leq e_i^S$ for $\exists i \in S$,
   otherwise, assign the $|S|+1$th lowest c_{uv} to $e_j^{S'}$

4. Let $e_i^{S'} = e_i^S$ if the edge corresponding to e_i^S still belongs to $E^{S'}$, otherwise, arbitrarily assigning one newly introduced c_{uv} to $e_i^{S'}$.

The construction starts with any *S* with *|S|=1*, $V_k \geq c_{oi}$ where $\{i\} \in S$.

Proposition 3.3 For each set $S \subseteq N$, the constraint

$$V_k \geq \sum_{i \in S} e_i^S y_{ik} \quad \text{for } k = 1,...,U \tag{3-17}$$

created by the construction procedure above are valid inequalities for the first model of the SDVRP.

Proof: The proposition is obviously true for any *S* with *|S|=1*. If we assume it is true for a set *S* and $i \in S$, we add another demand point *j* into *S* to have *S'*. We can have

$V_k \geq \sum_{i \in S} e_i^S y_{ik} + y_{jk} \min_{i \in S \cup \{0\}} c_{ij} \geq \sum_{i \in S} e_i^{S'} y_{ik} + e_j^{S'} y_{jk}$, because $e_i^S \geq e_i^{S'}$ and $\min_{i \in S \cup \{0\}} c_{ij} \geq e_j^{S'}$.

□

For instance, assuming we look at two demand points and the distances

between two points are $c_{01}=2$, $c_{02}=6$, $c_{23}=8$. The first valid inequalities are $V_k \geq 2y_1$ for $k = 1,...,U$ for set $S = \{1\}$. With the construction algorithm, after introducing demand point 2, the second valid inequality is $V_k \geq 2y_1 + 6y_2$ for $k = 1,...,U$ for set $S' = \{1,2\}$. Readers may wonder why we do not create the valid inequalities by arbitrarily assigning the first $|S|$ lowest c_{uv} $(u,v \in S \cup \{0\})$ to e_i for $i \in S$. A counterexample can be given for the previous example of the two demand points. The inequalities $V_k \geq 6y_1 + 2y_2$ for $k = 1,...,U$ for set $S' = \{1,2\}$ are not valid when $y_1=1$ and $y_2=0$. Therefore, the recursion is crucial to create these types of valid inequalities.

3.3 Numerical experiments

The data of the numerical experiment are from Lee et al. [7]. They use two methods to solve the SDVRP with small capacity: dynamic programming and pure MIP by directly using CPLEX. They compare the results of these two approaches to convince the advantage of their dynamic programming method. We will use the same data to do the numerical experiment on the desktop with PIII and 256M memory and have their outcome as our benchmark. In order to check the efficiency of the additional inequalities, we develop two Two-Stage methods, one (TS1) is only with triangular inequalities, and the other method (TS2) is with all inequalities we introduce in Section 3.2.

The capacity Q of the vehicle is assumed to be 1 without loss of generality, and

the location of the depot is set to be (0, 0). The positions and the demand quantity of the demand points are listed in the Table 3.1 and 3.2.

There are 9 layouts on the whole, and the numbers of demand points are 4, 5, and 7 for every 3 layouts respectively. The total demands for each layout were generated from 1.2 up to 9.6, with an incremental step of 0.4. The Computational results are in the Table 3.4 and 3.5 together with that of other methods.

Table 3.1: Geographic layouts for the problem instances

Code	N	Position						
N4L1	4	1(1,-3)	2(-6,-3)	3(-2,-8)	4 (0,-7)			
N4L2	4	1(7,7)	2(-2,0)	3(3,8)	4(-9,1)			
N4L3	4	1(1,-4)	2(3,1)	3(2,6)	4(8,-1)			
N5L1	5	1(2,7)	2(9,2)	3(9,-7)	4(-1,-7)	5(8,-7)		
N5L2	5	1(-10,-6)	2(-10,0)	3(-4,7)	4(1,1)	5(3,-10)		
N5L3	5	1(4,-8)	2(-2,5)	3(2,-6)	4(-4,-3)	5(1,2)		
N7L1	7	1(4,-6)	2(2,6)	3(7,7)	4(5,-5)	5(4,9)	6(-8,0)	7(5,-7)
N7L2	7	1(-10,-6)	2(-10,0)	3(-4,7)	4(1,1)	5(3,-10)	6(9,-10)	7(-1,4)
N7L3	7	1(4,-8)	2(-2,5)	3(2,-6)	4(-4,-3)	5(1,2)	6(6,-3)	7(-1,0)

These results shows that in the instances with small number of demand points (i.e., 4, 5 demand points and part of the 7 demand points), TS1, TS2 and the Dynamic programming based approach are much faster than direct MIP method, and the difference between the former three is very small. For larger size of the problems, TS2 is much faster than both TS1 and the DP approach.

Table 3.2: Demand Vectors

code	M=4				M=5					M=7						
	d1	d2	d3	d4	d1	d2	d3	d4	d5	d1	d2	d3	d4	d5	d6	d7
Q1	0.55	0.4	0.24	0.01	0.02	0.14	0.56	0.23	0.25	0.26	0.07	0.01	0.01	0.22	0.31	0.32
Q2	0.19	0.76	0.31	0.35	1.01	0.46	0.12	0.01	0.01	0.33	0.34	0.09	0.37	0.25	0.19	0.03
Q3	1.27	0.57	0.15	0.01	0.28	0.4	0.42	0.45	0.45	0.26	0.34	0.35	0.23	0.13	0.38	0.31
Q4	0.01	0.61	0.86	0.92	0.24	0.94	0.64	0.5	0.08	0.56	0.54	0.31	0.08	0.27	0.14	0.5
Q5	0.83	0.83	0.23	0.91	0.56	0.73	0.75	0.48	0.28	0.12	0.45	0.49	0.58	0.58	0.35	0.23
Q6	0.98	0.77	0.12	1.32	0.7	0.58	0.76	0.74	0.43	0.33	0.37	1.04	0.03	0.47	0.12	0.84
Q7	1.17	1.2	0.78	0.45	0.27	0.87	0.44	1.62	0.39	0.07	0.01	1.18	0.35	0.35	0.75	0.88
Q8	1.01	0.83	1.1	1.06	0.74	0.8	0.94	0.95	0.58	0.85	0.74	0.49	0.21	0.76	0.48	0.47
Q9	1.72	0.45	1.47	0.75	0.95	0.64	0.72	2.03	0.06	1.01	0.79	0.12	0.64	0.41	0.78	0.65
Q10	1.54	0.37	1.39	1.5	1.49	0.37	2.68	0.22	0.03	0.9	0.57	0.24	0.35	0.67	1.26	0.81
Q11	1.73	1.73	1.06	0.68	1.74	0.52	0.52	1.11	1.31	1.48	1.13	0.52	0.25	0.99	0.74	0.1
Q12	1.04	1.17	3.3	0.09	1.56	1.36	0.91	0.38	1.39	0.97	0.7	0.2	1.32	0.2	1.25	0.96
Q13	1.88	0.46	3.38	0.28	1.03	1.01	2.09	1.64	0.24	0.52	0.74	0.12	1.67	1.18	0.26	1.51
Q14	0.04	3.98	1.2	1.18	1.32	0.85	1.62	1.34	1.26	1.63	1.16	0.38	0.35	1.62	0.94	0.31
Q15	1.41	1.65	2	1.74	1.25	0.52	0.78	1.47	2.78	0.87	1.27	0.7	0.48	0.98	1.18	1.31
Q16	1.84	0.78	2.81	1.77	1.34	2.57	1.95	0.9	0.44	1.69	1.11	0.29	0.86	1.95	0.08	1.21
Q17	1.54	3.19	2.5	0.36	2.07	1.55	0.2	2.2	1.59	1.27	0.82	0.2	1.82	1.77	1.05	0.66
Q18	2.06	1.32	2.53	2.09	0.42	2.68	0.4	2.55	1.95	1.01	0.81	1.82	0.68	0.98	0.58	2.11
Q19	3.67	2.32	0.97	1.44	1.04	1.46	0.24	3.31	2.34	0.66	1.63	1.25	0.43	1.68	1.86	0.88
Q20	1.37	2.58	1.66	3.19	0.51	2.94	2.75	1.95	0.64	1.24	2.2	0.07	2.45	1.41	0.12	1.31
Q21	3.59	1.65	0.81	3.14	0.77	3.53	2.05	0.67	2.18	1.11	0.34	2.35	1.89	0.62	2.33	0.56
Q22	2.68	0.35	3.82	2.75	2.63	1.46	1.01	2.05	2.45	1.84	2.53	0.4	1.48	1.89	0.54	0.92

Table 3.3: CPU time and Cost from 4 methods for N=4

Code(Σdi)	N4L1					N4L2					N4L3				
	Cost	***TS1***	*TS2*	***SPA***	***MIP***	Cost	***TS1***	***TS2***	***SPA***	***MIP***	Cost	***TS1***	***TS2***	***SPA***	***MIP***
Q1(1.20)	28.67	<1.00	<1.00	<1.00	<1.00	40.69	<1.00	<1.00	<1.00	<1.00	32.34	<1.00	<1.00	<1.00	<1.00
Q2(1.61)	31.18	<1.00	<1.00	<1.00	<1.00	40.97	<1.00	<1.00	<1.00	<1.00	33.61	<1.00	<1.00	<1.00	<1.00
Q3(2.00)	28.96	<1.00	<1.00	<1.00	<1.00	56.79	<1.00	<1.00	<1.00	<1.00	36.79	<1.00	<1.00	<1.00	<1.00
Q4(2.40)	44.19	<1.00	<1.00	<1.00	1.34	44.68	<1.00	<1.00	<1.00	<1.00	44.68	<1.00	<1.00	<1.00	<1.00
Q5(2.80)	44.73	<1.00	<1.00	<1.00	2.19	59	<1.00	<1.00	<1.00	1.21	43.34	<1.00	<1.00	<1.00	<1.00
Q6(3.19)	51.22	<1.00	<1.00	<1.00	15.79	73.4	<1.00	<1.00	<1.00	6.4	54.3	<1.00	<1.00	<1.00	3.27
Q7(3.60)	55.38	<1.00	<1.00	<1.00	21.52	64.49	<1.00	<1.00	<1.00	0.5	47.51	<1.00	<1.00	<1.00	1.21
Q8(4.00)	59.45	<1.00	<1.00	<1.00	30.9	91.98	<1.00	<1.00	<1.00	29.1	64.79	<1.00	<1.00	<1.00	21.69
Q9(4.39)	64.5	<1.00	<1.00	<1.00	42.78	94.77	<1.00	<1.00	<1.00	22.22	59.85	<1.00	<1.00	<1.00	7.61
Q10(4.80)	71.17	<1.00	<1.00	1	77.71	95.69	<1.00	<1.00	<1.00	14.03	73.3	<1.00	<1.00	<1.00	58.62
Q11(5.20)	73.46	<1.00	<1.00	<1.00	296.01	85.56	<1.00	<1.00	1	27.24	65.4	<1.00	<1.00	<1.00	215.29
Q12(5.60)	91.85	<1.00	<1.00	1	709.83	112.05	<1.00	<1.00	<1.00	415.51	81.07	<1.00	<1.00	<1.00	268.06
Q13(6.00)	90.44	<1.00	<1.00	<1.00	282.3	125.14	<1.00	<1.00	<1.00	78.36	82.96	<1.00	<1.00	<1.00	50.03
Q14(6.40)	101.93	<1.00	<1.00	1	734.96	88.17	<1.00	<1.00	1	138.97	81.35	<1.00	<1.00	<1.00	563.25
Q15(6.80)	97.88	<1.00	<1.00	6	2904.6	117.51	<1.00	<1.00	<1.00	1107.67	85.27	<1.00	<1.00	<1.00	337.47
Q16(7.20)	103.54	<1.00	<1.00	1	N/A	131.08	<1.00	<1.00	<1.00	2027.33	93.01	<1.00	<1.00	<1.00	N/A
Q17(7.60)	113.91	<1.00	<1.00	1	N/A	120.05	<1.00	<1.00	1	4448.88	86.91	<1.00	<1.00	1	N/A
Q18(8.00)	109.68	<1.00	<1.00	1	N/A	150.98	<1.00	<1.00	1	N/A	108.13	<1.00	<1.00	<1.00	N/A
Q19(8.40)	103.54	<1.00	<1.00	<1.00	N/A	140.52	<1.00	<1.00	<1.00	N/A	91.02	<1.00	<1.00	<1.00	N/A
Q20(8.80)	126	<1.00	<1.00	1	N/A	153.29	<1.00	<1.00	1	N/A	116.79	<1.00	<1.00	2	N/A
Q21(9.20)	111.61	<1.00	<1.00	<1.00	N/A	172.74	<1.00	<1.00	<1.00	N/A	116.64	<1.00	<1.00	<1.00	N/A
Q22(9.60)	134.85	<1.00	<1.00	1	N/A	186.08	<1.00	<1.00	1	N/A	130.03	<1.00	<1.00	1	N/A

Table 3.4: CPU time and cost from 4 method s for N=5

Code(∑di)	N5L1					N5L2					N5L3				
	Cost	***TS1***	***TS2***	***SPA***	***MIP***	Cost	***TS1***	***TS2***	***SPA***	***MIP***	Cost	*TS1*	*TS2*	*SPA*	*MIPII*
Q1(1.20)	50.65	<1.00	<1.00	<1.00	<1.00	50.15	<1.00	<1.00	<1.00	<1.00	34.45	<1.00	<1.00	<1.00	<1.00
Q2(1.61)	56.52	<1.00	<1.00	<1.00	<1.00	71.81	<1.00	<1.00	<1.00	<1.00	51.09	<1.00	<1.00	<1.00	<1.00
Q3(2.00)	62.28	<1.00	<1.00	<1.00	<1.00	65.81	<1.00	<1.00	<1.00	<1.00	39.96	<1.00	<1.00	<1.00	<1.00
Q4(2.40)	67.14	<1.00	<1.00	<1.00	<1.00	69.25	<1.00	<1.00	<1.00	<1.00	42.6	<1.00	<1.00	<1.00	<1.00
Q5(2.80)	82.48	<1.00	<1.00	<1.00	1.31	74.66	<1.00	<1.00	1.00	<1.00	51.37	1.00	1.00	1.00	<1.00
Q6(3.19)	80.21	<1.00	<1.00	<1.00	5.56	82.32	<1.00	<1.00	1.00	7.58	55.49	1.00	1.00	<1.00	5.43
Q7(3.60)	83.24	<1.00	<1.00	<1.00	2.6	72.98	<1.00	<1.00	1.00	<1.00	53.34	1.00	1.00	<1.00	1.4
Q8(4.01)	91.21	<1.00	<1.00	<1.00	21.26	83.16	<1.00	<1.00	<1.00	4.79	55.78	1.00	1.00	<1.00	5.98
Q9(4.40)	89.76	<1.00	<1.00	<1.00	9.07	83.11	<1.00	<1.00	1.00	18.16	67.79	1.00	1.00	<1.00	7.8
Q10(4.79)	113.74	<1.00	<1.00	1.00	18.04	112.9	<1.00	<1.00	1.00	12.66	89.04	1.00	1.00	<1.00	43.38
Q11(5.20)	111.43	<1.00	<1.00	<1.00	460.18	124.02	<1.00	<1.00	2.00	2002.15	80.15	1.00	1.00	1.00	1294.38
Q12(5.60)	128.87	<1.00	<1.00	1.00	1846.23	131.03	<1.00	<1.00	1.00	1107.84	81.02	1.00	1.00	1.00	n/a
Q13(6.00)	142.6	<1.00	<1.00	<1.00	n/a	128.55	<1.00	<1.00	3.00	n/a	99.3	1.00	1.00	3.00	n/a
Q14(6.40)	142.91	<1.00	<1.00	1.00	n/a	136.15	<1.00	<1.00	3.00	n/a	88.18	1.00	1.00	2.00	n/a
Q15(6.80)	154.53	<1.00	<1.00	6.00	n/a	135.41	<1.00	<1.00	3.00	n/a	88.1	1.00	1.00	3.00	n/a
Q16(7.20)	154.88	<1.00	<1.00	1.00	n/a	146.94	<1.00	<1.00	2.00	n/a	107.12	1.00	1.00	2.00	n/a
Q17(7.60)	150.68	<1.00	<1.00	1.00	n/a	151.16	<1.00	<1.00	7.00	n/a	109.74	1.00	1.00	6.00	n/a
Q18(8.00)	168.08	<1.00	<1.00	1.00	n/a	144.29	1.00	1.00	4.00	n/a	89.35	1.00	1.00	4.00	n/a
Q19(8.40)	171.52	<1.00	<1.00	<1.00	n/a	150.9	1.00	1.00	9.00	n/a	102.95	3.00	3.00	8.00	n/a
Q20(8.80)	185.38	<1.00	<1.00	1.00	n/a	158.23	1.00	1.00	3.00	n/a	112.33	2.00	2.00	2.00	n/a
Q21(9.20)	202	<1.00	<1.00	<1.00	n/a	199.82	2.00	2.00	6.00	n/a	106.51	2.00	2.00	5.00	n/a
Q22(9.60)	191.02	<1.00	<1.00	1.00	n/a	198.8	2.00	2.00	11.00	n/a	123.49	4.00	2.00	10.00	n/a

Table 3.5: CPU time and cost from 4 methods for N=7

Code(Σdi)	N7L1					N7L2					N7L3				
	Cost	***TS1***	***TS2***	***SPA***	***MIP***	Cost	***TS1***	***TS2***	***SPA***	***MIP***	Cost	***TS1***	***TS2***	***SPA***	***MIPII***
Q1(1.20)	52. 33	<1.00	<1.00	1.	2	59.17	1	1	<1.00	3	38.49	1	1	<1.00	2
Q2(1.61)	54. 47	<1.00	<1.00	1	5	74. 88	1.	3	<1.00	8	39.21	5	4	<1.00	12
Q3(2.00)	57. 13	2	1	3	32	73.02	5	4	2	42	42.60	4	3	1	38
Q4(2.40)	77. 27	12	10	5	2310	81.14	7	4	5	2438	48.89	6	5	4	2167
Q5(2.80)	71. 86	7	5	6	5460	77.34	8	5	6	5210	45.95	6	4	7	4876
Q6(3.19)	88. 66	21	20	15	N/A	90.11	12	8	15	N/A	53.14	16	13	14	N/A
Q7(3.60)	85. 80	13	10	25	N/A	99.76	20	15	24	N/A	55.62	16	4	25	N/A
Q8(4.01)	90. 26	25	24	24	N/A	117.74	26	20	21	N/A	62.45	25	10	20	N/A
Q9(4.40)	93. 46	8	5	34	N/A	112.52	13	6	31	N/A	66.21	11	4	30	N/A
Q10(4.79)	107. 60	38	35	39	N/A	116.85	24	8	39	N/A	71.39	63	50	39	N/A
Q11(5.20)	101. 79	12	5	67	N/A	136.10	21	4	68	N/A	83.52	24	4	65	N/A
Q12(5.60)	120. 26	170	100	143	N/A	120.04	50	16	138	N/A	78.59	32	5	137	N/A
Q13(6.00)	128. 50	1340	50	73	N/A	114.39	780	30	77	N/A	61.92	520	10	77	N/A
Q14(6.40)	128. 15	2250	6	71	N/A	158.24	2139	5	69	N/A	91.37	1989	4	69	N/A
Q15(6.80)	133.13	N/A	8	270	N/A	161.42	N/A	53	255	N/A	86.84	N/A	35	239	N/A
Q16(7.20)	149. 70	N/A	64	200	N/A	161.46	N/A	10	186	N/A	90.37	N/A	48	187	N/A
Q17(7.60)	144. 97	N/A	15	300	N/A	161.91	N/A	13	300	N/A	93.89	N/A	30	296	N/A
Q18(8.00)	164. 07	N/A	127	755	N/A	154. 89	N/A	50	755	N/A	95.13	N/A	5	714	N/A
Q19(8.40)	153. 07	N/A	25	635	N/A	193. 60	N/A	200	667	N/A	99.02	N/A	10	615	N/A
Q20(8.80)	159. 19	N/A	81	161	N/A	164. 49	N/A	100	166	N/A	105.11	N/A	100	159	N/A
Q21(9.20)	180. 87	N/A	211	1331	N/A	188. 13	N/A	299	1375	N/A	125.13	N/A	325	1364	N/A
Q22(9.60)	175. 68	N/A	452	2780	N/A	196. 13	N/A	378	2888	N/A	116.02	N/A	521	2527	N/A

Table 3.6: New instance for N=15

Supplier	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(x, y)	(7, 7)	(-2,0)	(3 8)	(-9, -1)	(-2.3, 3)	(3.4, 5)	(-4, -1.5)	(1.2, -0.8)	(9, 3.5)	(-6.5, -1.2)	(3.4, -5)	(-4 –4)	(5.3, 3)	(-3 –5)	(6 –7)
a_i	0.35	0.19	0.42	0.34	0.25	0.31	0.37	0.21	0.55	0.16	0.18	0.24	0.31	0.54	0.32

For the biggest size of the problem with 9 suppliers below in Lee et al.’s paper,

their algorithm takes about 4 hrs and 48 minutes to obtain the solution. Our algorithm just takes about 2 minutes to achieve the same optimal solution.

Table 3.7: The case of N=9

Supplier	1	2	3	4	5	6	7	8	9
Position *(x, y)*	(4,-1)	(5, 3)	(-8, 5)	(-3, -2)	(5, 5)	(2, 2)	(9, -10)	(8, -10)	(-7, 2)
demand a_i	0.3	0.5	1.3	0.5	1.2	0.8	0.5	0.2	0.3

For our algorithm, the largest problem instance solved by TS2 within a reasonable time (not more than 3 hours) is 15 demand points by 5 vehicles. (See table 3.6), the optimal value is 100.99.

For almost all algorithms in the literature, the computational time is sensitive to the total demand for a given number of demand points because the number of vehicle index combination. Our algorithm's computational time doesn't explode with the number of total required vehicles because of the vehicle index assignment valid equalities/inequalities.

3.4 Remarks on the future work

The Vehicle Routing Problem with split delivery is an NP-hard problem. It is even harder than the classic Vehicle Routing Problem [2] because of more combinations in its structure. For the VRP, there are abundant papers in the literature to study on the exact algorithms, or the lower bound, or the efficient valid

inequalities developed for the polyhedron of the VRP. Sometimes, even those cuts from the TSP are borrowed to apply on the VRP due to the internal relationship between these two well-known problems. As to the Split Delivery Vehicle Routing Problem, the research work is far behind that of the VRP. Dror and Trudeau propose a branch-and-cut algorithm based on their work on the VRP with some inequalities, but they do not present the complete results of the instances. Belenguer, Martinez and Mota provide a cutting-plane method to obtain good lower bound of the Split Delivery Vehicle Routing Problem. In fact, both methods above are branch-and-cut, namely, they solve a sub-problem of the original one and add the violated constraints found to re-solve the problem to make the solution feasible. However, neither of the algorithms gives the final optimal integer solution to the problem. Lee et al. try another way to utilize dynamic programming to solve the SDVRP. Although they prove that they find a finite action space which is equivalent to the infinite action space of the SDVRP, the inherited weakness of the dynamic programming will incur "the dimension disaster" when the size of the problem increase, recalling the biggest size of the instance solved by the Lee et al.'s approach in a reasonable time is 9.

In this dissertation, we provide a Two-Stage exact algorithm to the Split Delivery Vehicle Routing Problem. This approach generalizes the classic cluster-first and routing-second heuristic algorithm to be an exact one. The technique we develop in this algorithm does make a bridge across the two sub-problems and let the first sub-problem have feedback from the second one for the first time. Therefore, due to

plenty of the similar models exist; we may apply this technique to such kind of problems.

Another contribution in this research work is the valid inequalities. In particular, the index assignment inequalities avoid a lot of replications because of identicalness of the vehicle in the fleet when we solve the problem. This scenario can be seen in other problems as well, for instance, a group of machines, aircrafts, or ships. Thus, we can apply the index fixing method to these problems to save computational time. We can also apply those valid inequalities to other type of vehicle routing problems.

Finally, we still need to focus on looking for more efficient valid inequalities for the Two-Stage exact algorithm since there is still distance between the result we obtain and our expectation. For example, we may explore to strengthen the triangular inequalities to exclude more routes from consideration.

CHAPTER IV

A BRANCH-AND-PRICE APPROACH TO THE SPLIT DELIVERY VEHICLE ROUTING PROBLEM

In this chapter, we study another type of the Split Delivery Vehicle Routing Problem: the number of vehicles in the fleet is a variable. This chapter consists of four sections. In Section 4.1, we formulate a column generation based split delivery vehicle routing problem. In Section 4.2, we propose a limit-search-tree-with-bound approach to the pricing problem. The branching strategy and the complete algorithm to the problem are provided in Section 4.3. The computational results and discussion on the problem are presented in Section 4.4.

4.1 Column generation based formulation of the SDVRP

In the Column Generation based formulation, each vehicle route is represented by a vector of a_j. The element a_{ij} of vector a_j is a continuous number and represents the demand picked up at demand point i by route a_j. Each column a_j has cost of c_j, representing the shortest distance traveled to visit all demand points in the route. Since there are numerous feasible routes, only a finite set of feasible routes is

chosen at the beginning and the restricted master problem (RMP) is constructed. A new route (column) with distance cost is generated by the sub-problem (pricing problem). Thus, the Column Generation based formulation of the SDVRP with the explicit pricing problem can be written as follows:

$$
\begin{aligned}
RMP \quad & \text{Min} \sum_{j\in\Omega} c_j x_j \\
& s.t. \quad \sum_{j\in\Omega} a_{ij} x_j = d_i, \quad i = 1,...,N; \qquad (4\text{-}1) \\
& \quad x_j = 0 \text{ or } 1 \ (\text{for all } j).
\end{aligned}
$$

Ω : the set of fesible routes,
x_j : a binary var iable, 1 if route j is used, 0 otherwise;
a_{ij} : amout picked up at demand point i on route j,
c_j : cost of route j, the shortest distance of the arcs making up the route.

The Pricing Problem:

$$
\begin{aligned}
& Min \ \sum_{i=0}^{N}\sum_{j=0}^{N} c_{ij}x_{ij} - \sum_{i=1}^{N} \pi_i a_i \\
& s.t. \ \sum_{j=0}^{N} x_{0j} = \sum_{j=0}^{N} x_{j0} = 1; \qquad (4\text{-}2) \\
& \quad \sum_{j=1}^{N} x_{ij} = \sum_{j=1}^{N} x_{ji} = y_i; \ \ i = 1,...N, \qquad (4\text{-}3) \\
& \quad a_i \le d_i y_i; \qquad i = 1,...N, \qquad (4\text{-}4) \\
& \quad \sum_{i=1}^{N} a_i \le Q; \qquad (4\text{-}5) \\
& \quad u_i - u_j + (N+1)x_{ij} \le N; \ \ i, j = 1,...N, \qquad (4\text{-}6)
\end{aligned}
$$

x_{ij} : 1, if the vehicle travels to demand point j from i directly;
0, otherwise. $i, j = 1,...,N$.

y_i : 1, if the demand point is visited by the vehicle; 0, otherwise.
$i = 1,...,N$.

a_i : load picked up at demand point i by the vehicle; $i = 1,...,N$.

u_i : dummy continous variables for subtour elimination;

π_i : the dual variable for ith constraint in the restricted master problem.

Constraints (4-2) and (4-3) are flow conservation constraints, while constraints (4-4) and (4-5) are supplier's demand constraints and vehicle constraints, respectively. Constraints (4-6) are sub-tour elimination constraints.

The column generation technique is effective to solve LP models with numerous variables (columns). Rather than using all variables of the LP model, the algorithm uses the pricing sub-problem to find the variables that have the lowest negative reduced cost and adds new columns to the master problem. When the objective function value of the pricing sub-problem is equal to or larger than 0, no new columns is generated, and thus the current solution to the master problem is the optimal solution to the LP relaxed RMP. Usually, the pricing problems are mixed integer-programming problems, such as knapsack problems in the cutting stock problem and the shortest path problem with resource constraints in the vehicle routing problem with time windows [18, 19, 24]. The optimal solution to the pricing problems may be obtained by certain exact solution methods. However, the pricing problem of a CVRP or an SDVRP is a capacitated prize-collecting Traveling Salesman Problem, which is an NP-hard problem **[**24**].** Therefore, it is difficult to

obtain the optimal solution even for medium size pricing problems. From the view of graph theory, the problem is defined on a complete and undirected graph; dynamic algorithm for the shortest path problem with resource constraints cannot work well. Agarwal et al.[24] use a nonlinear programming that is analogous to a knapsack problem to formulate the pricing sub-problem of their capacitated vehicle routing problem, and they present a linear function method to obtain a lower bound of the nonlinear objective function. Sierksma et al. [10] adopt the similar idea to work on their pricing problem for the routing helicopters for crew exchange problem. They define a subset S of the total N platforms (demand points) and calculate the TSP and the knapsack problem within the subset S separately. Since the number of subsets S is 2^{N-1}, they also provide a smart method that excludes a large amount of subsets from consideration.

4.2 A new algorithm to the pricing sub-problem

A limited-search-tree-with-bound algorithm is presented in this paper to solve the pricing problem of the column generation based formulation of the SDVRP. First, all demand points with nonzero π_i are sorted according to the non-increasing dual value π_i as candidate nodes in the search tree. The depot (point 0) represents the root node. Each node has two values: the unit reduced cost for the master problem if the associated demands are picked up without changing the basis of the master problem and its position. Since the number of demand points in one feasible branch

is typically small, such as six or less, an exact solution algorithm for the TSP is performed. A node is fathomed without further branching when it satisfies one of the following two criteria: 1) the accumulated load picked up at the current node exceeds the capacity Q of the vehicle; 2) the lower bound at the current node is larger than or equal to the current upper bound. The lower bound of a node is calculated based on the following lemma.

Lemma 4.1: Let k be the current node, S' be the set of the demand points searched before node k, $S' \subset S$, (S is the set of all nodes in the search tree). If node k does not violate the capacitated constraint (i.e. $\sum_{i \in S' \cup \{k\} \setminus \{0\}} \pi_i a_i \le Q$), its lower bound is $d_{r(S' \cup \{k\})} - \sum_{i \in S' \cup \{k\} \setminus \{0\}} \pi_i a_i - \pi_{k+1}(Q - \sum_{i \in S' \cup \{k\} \setminus \{0\}} a_i)$, where d_{r0} is the distance of the shortest tour to visit the set of demand points.

Proof: Since $\pi_i, i \in S$ are sorted on a non-increasing order and visiting more demand points will not decrease the length of the shortest tour visiting these points, the reduced cost of node k is $\sum_{i \in S' \cup \{k\} \setminus \{0\}} \pi_i a_i$, and the highest potential reduced cost of including other demand points is $\pi_{k+1}(Q - \sum_{i \in S' \cup \{k\} \setminus \{0\}} a_i)$. □

A node is fathomed if its lower bound is not smaller than the current upper bound. The following example is used to show how the algorithm works. We assume the dual prices from solving the restricted master problem in one iteration are $\pi_1 = 1.2$, $\pi_2 = 0$, $\pi_3 = 2.5$ $\pi_4 = 0.9$, $\pi_5 = 0$, and $\pi_6 = 1.7$, and are sorted as $(\pi_3, \pi_6, \pi_1, \pi_4)$ in a descending order, and their corresponding demands are a_3=0.24, a_6=0.35, a_1=0.13, and a_4=0.6. The pricing problem can be solved following the

procedures illustrated in Figure 4.1.

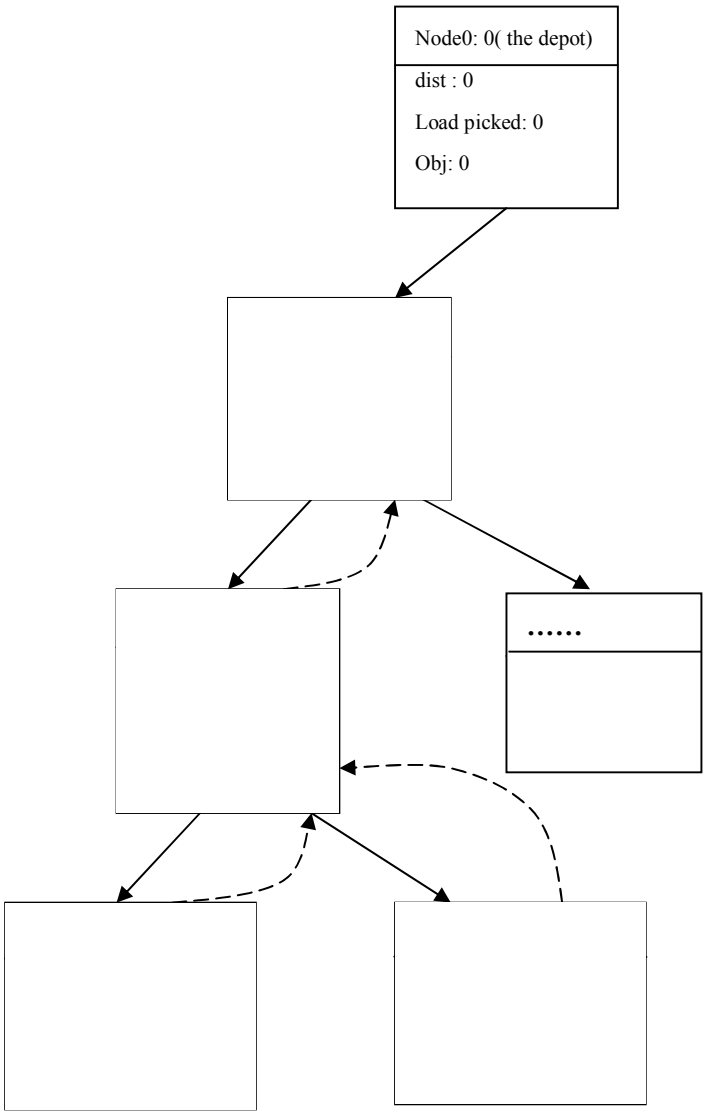

Figure 4.1: Limited-search-tree-with-bound procedure

A node in the search tree can be expressed by the following data structure:

```
Typedef struct node{
    int info;    /* the index of this node in the sorted demand point sequence;
    int childinfo;    /* the index of the next node generated by this node;
```

Set S'; /* the set of demand points traversed in the branch by this node;

int cut; /* indicate whether the node below this node will be cut or not;

double AccumLoad; /* the accumulated load picked up at the this node;

double obj; /* reduced cost of the current node, which is calculated as

$$d_{r(S'\cup\{\text{info}\})} - \sum_{i\in S'\cup\{\text{info}\}\setminus\{0\}} a_i * \pi_i$$

double dist; /* the distance of the shortest tour within the set $S'\cup\{\text{info}\}$;

double load; /* the load picked up at this node;

double lb; /* the lower bound of this node, which is calculated as stated in lemma 4.1;

struct node * parlink; /* a pointer points to the node that generates this node;

} NODE;

At the beginning, the upper bound of the pricing problem is set to ∞, and the lower bound of node 0 is set to $-\infty$. Node 0 is branched to node 1 which represents demand point 3. Travel distance $d_{r(0,3)}$ is $2c_{03}$, because its corresponding route is from the depot to demand point 3 and then back to the depot. Thus, the reduced cost is $d_{r(0,3)} - \pi_1 a_3$. Since the reduced cost of node 1 is $d_{r(0,3)} - \pi_1 a_3$, which is less than ∞, the upper bound of the searching tree is updated to $d_{r(0,3)} - \pi_1 a_3$. The lower bound of node 1 is $d_{r(0,3)} - \pi_1 a_3 - \pi_2 (Q - a_3)$. The cumulated load picked up so far is less than the vehicle capacity, because the pickup at this node is $a_3=0.24$. Since neither stop criterion satisfies, the search continues. The calculation at node 2 is

similar to that at node 1, and the search continues to move to node 3 by adding demand point 1. We assume at node 3 the present upper bound is less than the lower bound of node 3, node 3 is fathomed and no branches are created from node 3. The search is back to node 2 and then creates node 4 by adding demand point 4, which is just behind demand point 1 in the sorted sequence. At node 4, only a part of the demand of demand point 4 can be picked up since the vehicle is full. Therefore, node 4 is fathomed, and the search goes back to its parent node (node 2). If there are other candidate demand points in the queue, another new node will be created. Otherwise, node 2 will be fathomed. This procedure repeats until all the nodes are fathomed. At the end, the node providing the optimal solution (the upper bound) provides a new column (the loads at demand points in this node) with its cost coefficient of the upper bound for the RMP. For instance, if the upper bound is obtained at node 4, the optimal value of the pricing problem is $d_{r(0,3,6,4)} - \pi_3 a_3 - \pi_6 a_6 - \pi_4 (Q - a_3 - a_6)$ with the new column of $(0,0,a_3,Q-a_3-a_6,0,a_6)$.

The limited-search-tree-with-bound algorithm has several advantages over a general optimization solver for solving the original pricing problem. First, it decomposes the sub-problem into smaller TSP problems to avoid the memory overflow problem caused during solving large-size MIP problems. Secondly, this method generates not only the column with the highest reduced cost but also some other columns with negative reduced value. Adding these columns together with the

optimal value column into the master problem may reduce the total number of iterations and may be better than only adding one column each time provided by the simplex based integer programming solver. Thirdly, using branch-and-price algorithm requires the column with n*th* negative reduced cost at depth *n* in the branch-and-bound tree that is beyond the capability of some algorithms for this problem. Finally, this algorithm can avoid columns that are not allowed to produce if they are already in the column pool.

4.3 The branching scheme

Column generation technique is developed to solve the large size linear programming (LP). In order to obtain the feasible (optimal) integer solution, column generation should be integrated in a branch and bound framework, and it is called the branch-and-price algorithm **[**36**]**. This combination of column generation and branch-and-bound is not as easy as just solving a column generation problem followed by branch and bound to find an integer solution. There are fundamental difficulties in applying column generation techniq9ues for linear programming in integer programming solution methods [2]. First, conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem. Second, solving these LPs to optimality may not be efficient, in which case different rules will apply for managing the branch-and-price tree. Finally, it might not be possible to construct the optimal integer solution (even a

feasible integer solution) with the given columns from column generation steps. Therefore, new columns should still be generated after branching in order to obtain the integer solutions.

As M. Savelsbergh [22] has pointed out, branching strategies for 0-1 linear programs are based on fixing variables. There are two kinds of methods to perform variables fixing, one is a single variable fixing (variable dichotomy), the other is a set of variables fixing (GUB dichotomy). Their work indicates that fixing a single variable or fixing a set of variables in the standard formulation is equivalent to that in the disaggregated formulation, and the resulting branching scheme is compatible with the pricing problem.

For the SDVRP, we adopt the following branching scheme: if fractional number of the variables x_j for the Restricted Master Problem is obtained, then we set one of x_j to be zero, which means the corresponding route will not considered in the future. Otherwise, fixing x_j to 1 will require the route to be one of the candidate routes in the integer (optimal) solution, and those demand points whose load are fully picked up in this route are not allowed to be visited in the new routes generated by the pricing sub-problem.

The above branching strategy specifies how the current set of feasible solutions is to be divided into two smaller subsets. It does not specify how the sub-problem to be solved next is to be selected. The selection strategy we use here is depth-first search. This search is usually applied to obtain feasible solutions fast. Experience

shows that feasible solutions are more likely to be found deep in the tree than at nodes near the root. It is necessary to have a good feasible solution to be able to prune nodes and thus to reduce the size of the branch-and-bound tree.

4.4 Implementation and computational experiment

In this chapter, we propose a branch-and-price algorithm to solve the Split Delivery Vehicle Routing Problem. The algorithm is written in c with the CPLEX 9.0 Callable Library and run on a PC with 2.8GHz CPU, 512 MB of RAM.

The whole algorithm is as follows:

Step 1: Determine an initial feasible restricted master problem (RMP).

Step 2: Initialize a column pool with the existing columns in the RMP.

Step 3: Solve the current restricted master problem.

Step 4: Generate one or more columns with negative reduced costs that are not in the column pool by calling the limited-search-tree-with-bound routine. Add the column(s) to the restricted master problem and to the column pool. Go to step 3. If no such column can be generated, go to step 5.

Step 5: Get the optimal solution of the relaxation of the RMP, and initialize a root of a branch tree. Perform a proper branch scheme. In each node, repeat the procedures of step 1 to step 4 until the whole branch tree has been explored. Go to Step 6.

Step 6: Output the results.

This branch-and-price algorithm has been tested on a set of 11 instances from the TSPLIB, and a set of 14 randomly generated instances provided by Belenguer et al. [10]. The vehicle capacity is always Q=160, and the demands are randomly generated within an interval expressed as a function of Q. Computational results are reported in Tables 4.1 and 4.2, compared with the results obtained by Belenguer et al.'s cutting-plane algorithm. The following columns summarize the results of both algorithms:

LB: the lower bound,

UB: the upper bound,

GAP: the percentage of (UB-LB)/UB.

K, K', and K'' represent the number of vehicles needed in the VRP, the instances of the SDVRP in the Belenguer et al.'s paper and in our method, respectively.

As to "Ratio", it is calculated by d(V)/KQ, where d(V) is the total demand and KQ is the total capacity. "Ratio" reflects how difficult an instance is.

Observed from Table 4.1, about 50% of results obtained by the branch-and-price algorithm have a better lower bound. In Table 4.2, 6 out of 8 instances have better outcomes both in lower bound and upper bound (feasible integer solution). Belenguer et al. argue that the instances in Table 4.1 seem to be more difficult than that in Table 4.2. But according to the experience of our algorithm, we have the opposite conclusion.

Table 4.1: Computational results on some TSPLIB instances

Method				cutting-plane algorithm			branch-and-price algorithm		
Instance	K	Q	Ratio	LB	UB	Gap	LB	UB	Gap
Eil22	4	6,000	0.94	375.0	375	0.0	373.6	376	0.6
Eil23	3	4,500	0.75	569.0	569	0.0	564.3	608	7.2
Eil30	3	4,500	0.94	508.0	510	0.39	507.2	515.3	1.6
Eil33	4	8,000	0.92	833.0	835	0.24	830.2	873.4	4.9
Eil51	5	160	0.97	511.6	521	1.81	507.6	558.5	9.1
EilA76	10	140	0.97	782.7	832	5.92	800.3	900.7	11.1
EilB76	14	100	0.97	937.5	1,023	8.36	965.7	1163.1	17.0
EilC76	8	180	0.95	706.0	735	3.94	711.2	809.3	12.1
EilD76	7	220	0.89	659.4	683	3.45	652.3	768.8	15.2
EilA101	8	200	0.91	793.5	817	2.88	797.5	910.2	12.4
EilB101	14	112	0.93	1,005.9	1,077	6.61	1013.9	1174.1	13.6

Table 4.2: Computational results of the two algorithms on randomly generated instances

Method			cutting-plane algorithm				branch-and-price algorithm			
Instance	*K*	Ratio	*K'*	LB	UB	Gap	*K*"	LB	UB	Gap
S51D1	3	0.84	3	454	458	0.87	3	449.9	513.9	12.5
S51D2	9	0.98	9	676.6	726	6.80	9	556.7	1296.5	57.0
S51D3	15	0.95	15	905.2	972	6.87	15	956	986	3.14
S51D4	30	0.99	27	1,521	1,677	9.32	29	1623	1654	1.91
S51D5	26	0.99	23	1,273	1,440	11.61	25	1416	1434	1.27
S51D6	50	0.98	41	2,113	2,327	9.20	41	2270	2316	2.03
S76D4	40	0.97	37	2,012	2,257	10.87	39	2178	2205	1.24

This contradiction is due to the principle of the two algorithms. Belenguer et al.'s algorithm is more inclined to solving the TSP, which means it works well when the capacity of vehicle is large and the number of vehicle needed is small (less than 6). This kind of instances in Table 4.2 is more like the UPS or FedEx routing problem. When the number of vehicles needed in the problem is larger than 6, their algorithm cannot obtain good results as previous ones. This type of instances is more

like truckload routing problems and our algorithm seems to be good at it. Therefore, the branch-and-price algorithm is competitive to the cutting-plane algorithm, and is promising in the instance where the number of vehicles needed is large.

CHAPTER V

CONCLUSION

This chapter consists of two sections. Section 5.1 proposes the contribution of the research work in this dissertation. In Section 5.2, we discuss the future work associated with our current study.

5.1 Contribution

In this dissertation, we examine the Split Delivery Vehicle Routing Problem (SDVRP), which is a relaxed version of the classic Vehicle Routing Problem (VRP). This problem was first introduced by Dror and Trudeau over a decade ago. Like its parental problem, the SDVRP is an NP-hard problem, even "harder" than the VRP.

There are two cases in the Split Delivery Vehicle Routing Problem. One is the number of vehicles in the fleet is a fixed number as the minimal required number of vehicles, while in the other case the vehicle number is a variable. In the literature, Dror and Trudeau [1, 8, 9], Sierksma and Tijssen [15] try to solve the SDVRP with a various number of vehicles and focus on minimizing the total travel distance, while Belenguer et al., Lee et al. cope with the Split Delivery Vehicle Routing Problem with the fixed number of vehicles [9, 10].

We study both scenarios of the SDVRP in this dissertation. For the SDVRP with

a fixed number of vehicles, we provide a cutting-plane based exact method called Two-Stage algorithm where the SDVRP is decomposed into two phases of clustering and routing. At the first stage, an assignment problem is resolved to attain clusters that cover all demand points and to obtain the initial lower bound for the whole problem; at the second stage, the minimal travel distance in each cluster is calculated as a classic Traveling Salesman Problem (TSP) to obtain the upper bound. We find a way to make these two phases to communicate mutually for the first time. This method yields a new exact approach to the Split Delivery Vehicle Routing Problem rather than the heuristic one in the literature. Furthermore, we develop a family of efficient valid inequalities to improve the performance of the algorithm significantly. For instance, in order to avoid the replication in the process of finding the optimal solution, we design an index assignment method. This method is a generalization of the variable fixing method which is mentioned in Dror and Trudeau's paper [1].

We consider another scenario when the number of the vehicles is a variable in this dissertation as well. A column generation based branch-and-price algorithm is presented. Although this methodology is applied comprehensively, it is the first time to use this approach in this problem. We also develop a limit-search-tree-with-bound algorithm to solve the sub-problem in the column generation method. This sub-problem itself is an NP-hard problem, which is called capacitated prize-collecting traveling salesman problem. The algorithm we provide has several advantages over a general optimization solver, e.g., CPLEX.

The computational results indicate that both approaches are competitive to those in the literature.

5.2 Future work

In the future, we may extend the current work by the following two ways. First, we can do some research work to deepen and enrich the present algorithms for the Split Delivery Vehicle Routing Problem. For instance, for the algorithm provided by Belenguer et al., new facet-defining inequalities that can strengthen the formulation have not been used. Therefore, the results could be improved if we design identification procedures that could be added to the algorithm.

Moreover, the heuristics in Chapter IV can be improved to produce better lower bound as well, and the information provided by such a good lower bound may be used to design new heuristic algorithms to obtain better upper bound. In fact, exploration on the efficient valid inequalities is also required in our Two-Stage exact algorithm. For example, we may try to improve the triangular inequalities to exclude more routes from consideration. For the branch-and-price approach, the final success of this approach depends heavily on the resolving of the sub-problem efficiently.

We may also apply the techniques and ideas used in these algorithms to other fields. Lee et al. present a dynamic programming based exact algorithm for the Split Delivery Vehicle Routing Problem. In their research, they found that although the most natural such formulation for the SDVRP contains an uncountable infinite state

space, it is possible to modify the formulation to obtain a dynamic programming with a finite state space. This technique on the reduction of action space is inspiring, and we may apply it to other actual problems. In this dissertation, we develop an idea to build a bridge across the two sub-problems and let the first sub-problem have feedback from the second one for the first time in our Two-Stage algorithm, which makes the approach to be exact rather than heuristic. In fact, there are plenty of problems that can be decomposed into several phases. Therefore, we may try to apply this technique to these problems. Another technique we present is the index assignment inequalities. This class of inequalities avoids a lot of replications due to identicalness of the vehicle in the fleet when we solve the problem. The scenario occurs in other problems as well, for instance, a group of machines, aircrafts, or ships. Thus, we can apply the index fixing method to these problems to save a lot of computational time. We may lend those valid inequalities to other type of vehicle routing problems as well.

REFERENCES

[1] Dror, M., G. Laporte, and P. Trudeau, Vehicle routing with split deliveries. Discrete Appl. Math. 1994; 50: 239-254.

[2] Altinkemer, K. and B. Gavish, Heuristics for unequal weight delivery problems with a fixed error guarantee. Oper. Res. Lett. 1987; 6: 149-158.

[3] Altinkemer, K. and B. Gavish, Heuristics for delivery problems with constant error guarantees. Trans. Sci. 1990; 24:294-297.

[4] Christofides, N. et al. The Traveling Salesman Problem: A guided Tour of Combinatorial Optimization (Wiley, Chichster, UK, 1985) 361-401.

[5] Laporte, G., The vehicle routing problem: an overview of exact and approximate algorithms, J. Oper. Res. 1992; 59: 509-514.

[6] Laporte, G. and Y. Nobert, A branch and bound algorithm for the capacitated vehicle routing problem, OR Spektrum 1983; 5: 77-85.

[7] Laporte, G. and Y. Nobert, Exact algorithms for the vehicle routing problem, Surveys in Combinatorial Optimization, Annals of Discrete Mathematics, 1987; 31: 147-185.

[8] Dror, M and P. Trudeau, Split delivery routing. Naval Re. Logist. 1990; 37: 383-402.

[9] Dror, M. and P. Trudeau, Savings by split delivery routing. Trans. Sci. 1989; 23: 141-145.

[10] Belenguer, J. M., M. C. Martinez and E. Mota, A lower bound for the split delivery vehicle routing problem. Oper. Res. 2000; 48: 801-10.

[11] Lee, C. G., M. Epelman, and C. C. White C-C, A shortest path approach to the Multiple-vehicle routing with split picks-up. Submitted to Transportation

Science. 2003.

[12] Frizzell, P.W. and J. W. Giffin, The bounded split delivery vehicle routing problem with grid networks distances. Asia Pacific J. Oper. Res. 1992; 9:101-116.

[13] Frizzell, P.W. and J. W. Giffin, The split delivery vehicle scheduling problem with time windows and grid network distance. Comput. Oper. Res. 1995; 22: 655-667.

[14] Mullaseril, A., M. Dror, and J. Leung, Split-delivery routing heuristics in livestock feed distribution. J. Oper. Res. Soc. 1997; 48: 107-16.

[15] Sieksma G., and G. A. Tijssen, Routing helicopters for crew exchanges on off-shore locations. Ann. Oper. Res. 1998; 76: 261-286.

[16] Ford, L. and D. Fulkerson, A suggested computation for maximal multicommodity network flows. Management Sci. 1958; 5:97-101.

[17] Dantzig, G. and P. Wolfe, Decomposition principle for linear programs. Oper. Res. 1960; 8:101-111.

[18] Gilmore, P.C. and R. E. Gomory, A linear programming approach to the cutting stock problem. Oper. Res. 1961; 9: 849-859.

[19] Gilmore, P.C. and R. E. Gomory, A linear programming approach to the cutting stock problem-Part II. Oper. Res. 1963; 11: 863-888.

[20] Desrochers, M., J. Desrosiers and M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. 1992; 40:342-354.

[21] Lubbecke, M. and J. Desrosiers, Selected topic in column generation. Technique Report, Braunschweig University of Technology, December, 2002.

[22] Savelsbergh, M.W.P., A branch and price algorithm for the generalized assignment problem. Oper. Res. 1997; 45: 831-841.

[23] Agarwal, Y., K. Mathur and H. M. Salkin, A set-partitioning-based exact

algorithm for the vehicle routing problem. Networks, 1989; 19: 731-749.

[24] Desrosiers, J., F. Soumis and M. Desrochers, Routing with time windows by column generation. Networks, 1984; 14: 545-565.

[25] Sol, M., Column generation techniques for pickup and delivery problem. PhD thesis, Eindhoven University of Technology.

[26] Forrest, J. and D. Goldfarb, Steepest-edge simplex algorithms for linear programming. Math. Programming 1992; 57:341-374.

[27] Goldfarb, D. and J. Reid, A practicable steepest-edge simplex algorithm. Math. Programming 1977;12:361-371.

[28] Harris, P., Pivot selection methods of the Devex LP code. Math. Programming 1973;5:1-28.

[29] Clarke, G. and J. Wright, Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. 1964; 12: 568-581.

[30] Barnhart, N. L. et al., Flight string models for aircraft fleeting and routing. Trans. Sci. 1998; 32: 208-220.

[31] Valerio de Carvalho, J. M., Exact solution of bin-packing problems using column generation and branch-and-bound. Ann. Oper. Res. 1999; 86:629-659.

[32] Mehrotra, A. and M. A. Trick, A column generation approach for graph coloring. INFORMS J. Comput. 1996; 8(4): 344-354.

[33] Erdmanm, A. et al, Modeling and solving an airline scheduling generation problem. Ann. Oper. Res. 2001; 107: 117-142.

[34] Marsten, R.E., W.W. Hogan and J.W. Blankenship, The boxstep method for large-scale optimization. Oper. Res. 1975; 23: 389-405.

[35] Merle, O. et al., Stabilized column generation. Discrete Math. 1999; 194: 229-237.

[36] Barnhart, C., Branch-and-price: column generation for solving huge integer programs. Oper. Res. 2000; 46: 316-329.

[37] Vanderbeck, F., On Dantzig-Wolfe decomposition in integer programming

and ways to perform branching in a branch-and-price algorithm. Oper. Res. 2000; 48:111-128.

[38] Bixby, A., C. Coullard and D. Simchi-Levi, The Capacitated Prize-Collecting Traveling Salesman Problem. International Symposium on Mathematical Programming Lausanne, EPFL, 1997.

[39] Ben A., H., Stabilization algorithms of column generation. Ph.D. Dissertation Ecol Polytechnique de Montreal, 2002.

[40] Anbil, R., J. Forrest and W. Pulleyblank, Column generation and the airline crew pairing problem. ICM, 1998; 3:677-686.

[41] Vance, P., Branch-and-Price algorithms for the one-dimensional cutting stock problem. Comput. Optim. Appl. 1998; 9:211-228.