Optimal Shipping Decisions in an Airfreight Forwarding Network

by

Zichao Li

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2012

Abstract

This thesis explores three consolidation problems derived from the daily operations of major international airfreight forwarders.

First, we study the freight forwarder's unsplittable shipment planning problem in an airfreight forwarding network where a set of cargo shipments have to be transported to given destinations. We provide mixed integer programming formulations that use piecewise-linear cargo rates and account for volume and weight constraints, flight departure/arrival times, as well as shipment-ready times. After exploring the solution of such models using CPLEX, we devise two solution methodologies to handle large problem sizes. The first is based on Lagrangian relaxation, where the problems decompose into a set of knapsack problems and a set of network flow problems. The second is a local branching heuristic that combines branching ideas and local search. The two approaches show promising results in providing good quality heuristic solutions within reasonable computational times, for difficult and large shipment consolidation problems.

Second, we further explore the freight forwarder's shipment planning problem with a different type of discount structure - the system-wide discount. The forwarder's cost associated with one flight depends not only on the quantity of freight assigned to that flight, but also on the total freight assigned to other flights operated by the same carrier. We propose a multi-commodity flow formulation that takes shipment volume and over-declaration into account, and solve it through a Lagrangian relaxation approach. We also model the "double-discount" scheme that incorporates both the common flight-leg discount (the one used in the unsplittable shipment problem) and the system-wide discount offered by cargo airlines.

Finally, we focus on palletized loading using unit loading devices (ULDs) with pivots, which is different from what we assumed in the previous two research problems. In the international air cargo business, shipments are usually consolidated into containers; those are the ULDs. A ULD is charged depending on whether the total weight exceeds a certain threshold, called the *pivot weight*. Shipments are charged the *under-pivot rate* up to the pivot weight. Additional weight is charged at the *over-pivot* rate. This scheme is adopted for safety reasons to avoid the ULD overloading. We propose three solution methodologies for the air-cargo consolidation problem under the pivot-weight (ACPW), namely: an exact solution approach based on branch-and-price, a best fit decreasing loading heuristic, and an extended local branching. We found superior computational performance with a combination of the multi-level variables and a relaxation-induced neighborhood search for local branching.

Acknowledgements

I would like to thank my advisors Dr. Jim Bookbinder and Dr. Samir Elhedhli for the unique perspective and expertise they each brought to this thesis and my graduate studies. I valued both greatly. I could not have asked for more from doing research with the two of them; it simply has been a lot of challenge and progress. I would especially like to thank both of them for their involvement and patience with my maturation as a researcher and writer.

My gratitude also goes to the faculty of the Department of Management Sciences for providing me with the foundations for my Ph.D. research. In the meantime, I would thank Dr. Teodor G. Crainic, Dr. Liping Fu, Dr. Fatma Gzara and Dr. Elizabeth Jewkes for their time and effort in serving on my Ph.D. committee.

I would like to thank all my friends in Kitchener-Waterloo who made my stay in Canada so memorable, especially my great fellow students Bissan, Da, Eman, Emre, Ginger and Joe, who provided comprehensive help in research, immigration and daily living. My special thanks to my officemate Mey, who shared a series of coincidental footsteps with me in Singapore, United States and Canada. I also benefited from the facilities of WATMIMS (Waterloo Management of Integrated Manufacturing Systems), as well as first-class services from our department's staff: Bev, Carol, Kathy and Wendy.

Finally, I am forever indebted to my parents for their unconditional love and support, and for the sacrifices they have always made to see me realize my dreams in life.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivations

This dissertation aims to investigate the cost saving opportunities that come with air cargo transportation. "We" (you, the reader and myself, the author) will focus the study from a freight forwarder's perspective.

International air cargo is an operation-intensive industry, involving complex procedures and many players (Huang and Chi, 2007). The main players in the air cargo supply chains are: airlines, freight forwarders, and shippers. The shippers send their loads to freight forwarders, and forwarders tender the freight to airlines. Freight forwarders satisfy the demands of shippers by securing cargo capacity from the airlines. Generally, airlines offer cargo space in two stages. In the first, a few months prior to a season, freight forwarders bid for upcoming cargo space. The cargo capacity committed during this bidding process is called *allotted capacity*. Airlines usually allocate some of the remaining space to contracts, which is the space reserved for large customers at an offered price. The remaining cargo space, the capacity available for free sale, is open for booking in the second stage, within a few days before the flight departs.

Forwarders are important players in the international supply chain. They do not own airplanes, but rather rely on airlines to transport their cargo. A limited number of international forwarders own some airplanes, but still heavily depend on external airlines. Customers typically get in touch with forwarders instead of airlines: Airlines have very limited number of destinations with larger capacities, while forwarders can reach more destinations and have access to greater capacities. Freight forwarders can provide the customer with door-to-door delivery. For certain destinations, a combination of different transportation modes results in much lower transportation costs, adding to the advantage of using a freight forwarder.

Airfreight forwarders need to meet the customer's transportation requirements while minimizing the expense charged by the airlines. However, the air cargo rate structure is very complicated: In the most basic *flight-leg discount*, the cost (or discount) on each flight leg depends upon both shipment weight and volume. This makes the consolidation problem a difficult one for the airfreight forwarder. A given shipment will be routed through various stations in the airfreight forwarding network before reaching its final destination. This routing decision has to be made in consideration of possible consolidation with any other shipments in the network. This leads to the consolidation problem faced by freight forwarders, as they try to maximize their revenue from customers while being charged for a minimum payload by the carriers. Other than the flight-leg discount, there are various additional incentives offered by cargo airlines to attract freight forwarders. We are motivated to find out the optimal decisions for forwarders in each of the three problems for forwarders below.

1.1 Research Problems

1.1.1 Unsplittable Network Consolidation with Volume Weight

A freight forwarder manages the shipping process for a customer (Leung et al., 2009). The major decision in that process is the assignment of shipments to flight legs. In Chapter 3, we focus on the freight forwarding decision problem at a tradelane level. A *tradelane* refers to a set of origins and destinations which are geographically pairwise adjacent. Each tradelane is part of a freight forwarder's global network that consists of a set of airports classified as exporting stations, exporting hubs, importing hubs and importing stations. For example, in the Sino-US tradelane (Fig 1.1), the China operations manager needs to decide on the routing of export shipments from China to the US. Suppose we have a number of air shipments in Chongqing (CHQ), China to be delivered to Denver (DEN), Colorado, USA. The shipments have to be first consolidated at major exporting hubs in China, namely Beijing (PEK), Shanghai (PVG), Guangzhou (CAN) or Hongkong (HKG). Then, the consolidated shipments are transported to US importing hubs such as San Francisco (SFO), Los Angeles (LAX) or Chicago (ORD), where shipments are de-consolidated and transferred to the final destination Denver. There are given capacities between outbound hubs in China and inbound hubs in the United States.

The leg between the cargo origin and the exporting hub in China is often called the *Feed Leg*. *Defeed Leg* refers to the leg between a US gateway hub and a final destination in the US. The leg between hubs in China and those in the US is called the *Major International Line-Haul* (Fig. 1.1). The result is a layered network similar to that described in Balakrishnan and Graves (1989) with nodes classified into four types: origin nodes, export hub nodes, import hub nodes and destination nodes.

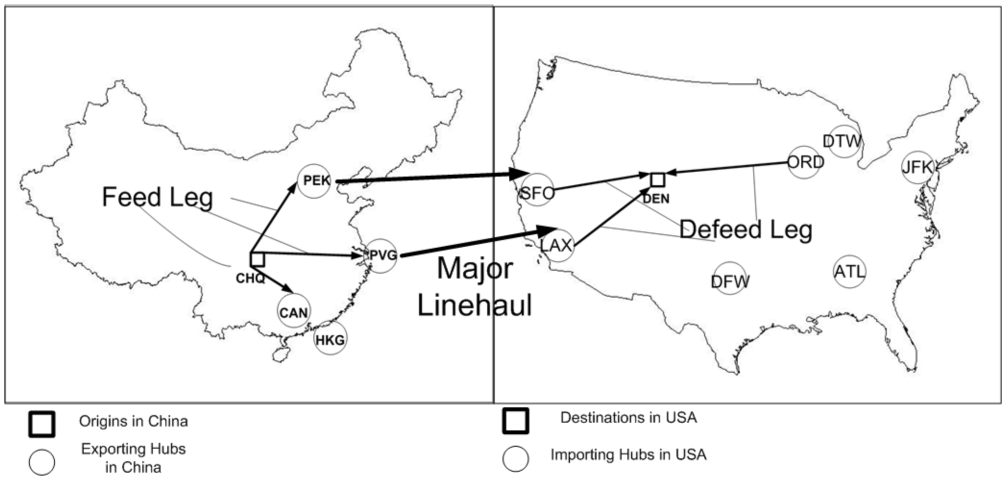


Figure 1.1: International Airfreight Network Structure

Although the decisions may appear similar, a cargo airline and a freight forwarder have two different objectives for a given tradelane network. From the airline's perspective, the objective is to find the minimum-cost route for a set of shipments by taking advantage of economies of scale, while having to consider loading constraints and cargo allocation constraints. From the freight forwarder's perspective, the objective is to minimize the cost through balancing chargeable weight and volume weight, exploiting quantity discounts, avoiding late penalty charges, and reserving the proper space with the cargo airlines. The last aspect, reservation of space, requires better forecasting, which will not be covered here.

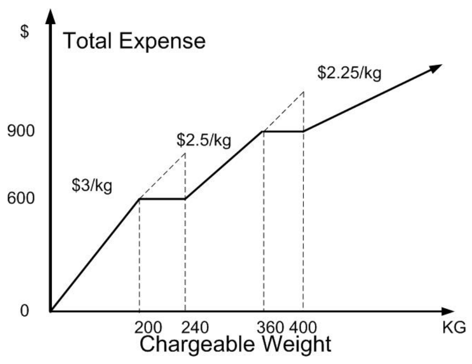
Balancing Gross Weight and Volume Weight

The air cargo rating system takes into account the shipment volume in addition to its gross weight. According to Huang and Chi (2007), the *volumetric weight* is obtained by dividing the shipment volume in cm^3 by a constant, 6000 cm^3/kg, currently adopted by the industry. The *chargeable weight* is the greater of the gross weight and the volumetric weight. Therefore, airfreight forwarders have an incentive to combine smaller loads from different shippers into a larger and consolidated shipment. The overall chargeable weight of a consolidated shipment containing low- and high-density items is less than the sum of individual chargeable weights.

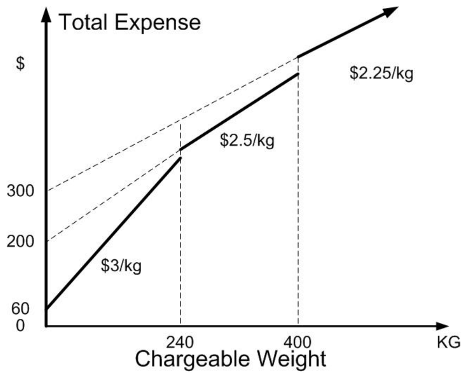
Exploiting Quantity Discounts

There are two ways to take advantage of quantity discounts. In the first, as the unit cost on each flight is divided into several decreasing price segments (Fig. 1.2(a)), forwarders try to consolidate as much as they can on a single flight to obtain a lower price. The rates in Figure 1.2(a) are \$3/kg, \$2.50/kg, and \$2.25/kg for shipments less than 240 kg, between 240 kg and 400 kg, and greater than 400 kg respectively. In the air cargo industry, the weights of 240 kg and 400 kg in this example are called *weight break points*. Obviously, if a forwarder can consolidate shipments where the overall weight exceeds 400kg, the total cost per unit shipped would be minimized. We note that the quantity discounts used in freight forwarding industry are typically of the "all unit" type; an incremental discount is rarely seen.

Secondly, between these weight breakpoints, a forwarder may *over-declare* a quantity to take advantage of the next discount (e.g. Carter et al. (1995)). As in Figure 1.2(a), if a shipment with weight between 200 and 240 kg is declared to be 240 kg, it would be charged $\$2.50/kg$. This is a saving compared to paying $\$3/kg$ for the actual chargeable weight. A similar situation applies to consolidated shipments in the range of 360 kg to 400 kg. Therefore, for loads between certain weights, it is desired to over-declare them to the minimum quantity of the next weight range. This practice of over-declaration, when favorable, is sometimes referred to as making use of the "bumping clause", whereby the shipment is pushed into the next higher weight range.



(a) Without Fixed Charge



(b) With Fixed Charge

Figure 1.2: Airfreight Discount Schedule with and without a Fixed Charge

Moreover, it is quite common to have a fixed cost, called the "document cost" or "consolidation cost", for each weight range in the cost structure. Such a fixed cost does not favor over-declaration (See Fig. 1.2(b)). In contrast, for the same case without the fixed cost, we could create two regions $(200, 240)$ and $(360, 400)$, for *over-declaration* as shown in Figure 1.2(a). Scenario 1.2(b) resembles a production system with setup cost. In practice, the document cost is usually very small compared to the shipment costs and can be ignored, making figure 2-(a) more popular for the air cargo industry. Hence, we assume this fixed charge is 0 throughout Chapter 3.

In this thesis, the goal is to build and optimize decision models that address six important practical characteristics. The models incorporate as appropriate, one or more of the following characteristics:

1. Multiple origins and multiple destinations.
2. A capacity constraint on flight legs or network arcs.
3. Economies-of-scale on each flight leg.
4. Shipment volume alongside shipment weight.
5. Possible over-declaration to the next weight range.
6. Flight departure/arrival time and shipment ready/delivery time.

1.1.2 Consolidation Problem Under System-wide Discount

Apart from the widely known flight-leg discount, regional and passenger airlines typically offer a system-wide discount, if a freight forwarder exceeds a specific aggregate quantity on all flights operated by that carrier within a certain period of time. Thus, the forwarder's cost associated with one flight not only depends on the quantity of freight assigned to that flight, but also on the total freight assigned to other flights operated by that carrier (Cohn et al., 2008). This type of discount scheme is typically offered by a passenger airline that operates multiple daily flights on certain routes, and carries cargo in its lower cabin. Due to limited belly capacity, such airlines offer the discount to attract high-volume customers. Forwarders that handle large numbers of small parcels are the greatest beneficiaries of this discount scheme, as they fail to qualify for volume discounts on conventional cargo airlines.

Due to the seasonality and trade imbalance, cargo airlines are increasingly motivated to offer forwarders a system-wide discount in low season and on less-utilized routes. Unlike the piece-wise linear "bumping-clause" discount scheme (Higginson and Bookbinder (1994); Croxton et al. (2003); Huang and Chi (2007); Chang (2008)), which is popular in the airfreight industry, there is very little literature that focuses on the system-wide problem. Cohn et al. (2008) were the first to study this case, and model it as a network flow problem where arc costs depend not only on the flow on that arc, but also on the flow on all arcs associated with the same carrier. Therefore, the cost has two components: a per-unit base cost, and a discount factor that depends on the quantity of freight flowing over all arcs associated with the same carrier. Cohn et al. (2008), however, do not account for certain practical features of the cargo business, namely the "unsplittable" requirement (a given shipment cannot be divided), shipment volume alongside weight, and "over-declaration". We will incorporate those features into a mathematical programming formulation and propose solution methodologies that are capable of solving large-size problems.

1.1.3 Pivot-Weight Scheme

The operations of an airfreight forwarder include making capacity reservations with cargo airlines, consolidating shipments, tendering freight to the airline, and breaking bulk for the final delivery at the destination. In the first stage, large forwarders sometimes reserve a desired number of *containers* from airlines (instead of reserving in terms of chargeable weight). These containers are called Unit Load Devices (ULDs) in the industry. Each airline offers several types of containers. They differ in fixed reservation charges, pivot weights, unit pivot costs, maximum weights and over-pivot rates.

The *pivot weight* is a weight threshold of a ULD, under which the cargo is charged at a rate of unit pivot cost. Any weight that falls between the pivot weight and the maximum capacity is charged at a special unit rate higher than the pivot rate, called the over-pivot rate. In addition, there is a fixed reservation cost for each reserved ULD. Faced with this pricing scheme, a forwarder is interested in finding the optimal consolidation decisions to minimize total cost. This problem is commonly encountered by large freight forwarders and airline operators.

Note that the pivot-weight scheme discussed here is not a discount. Airlines price in this way to prevent shippers from overloading ULDs. The over-pivot cost can be seen as an incremental penalty, rather than a discount. The well-known "bumping-clause", as in Bookbinder and Higginson (2002), whereby it can be advantageous to over-declare the total weight dispatched, is used more for general cargo (goods that are not containerized). Rather the pivot-weight scheme is suitable for shipments consolidated in ULDs.

Li et al. (2009) give a detailed description of the problem and propose a large-scale neighborhood search heuristic to solve this problem. However, one of the constraints presented by those authors is redundant and can be discarded to simplify the problem decomposition. Moreover, Li et al. (2009) do not furnish any solution methodologies that can deliver the solution with an exact method. In this problem, we aim to address the deficiencies in their paper by providing three solution methodologies for ACPW (the Airfreight Consolidation Problem under the Pivot Weight).

1.2 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, we review the existing models on operational planning and consolidation in transportation, as well as the corresponding solution methodologies to solve those models. In Chapter 3, we discuss the network consolidation that incorporates volume weight, over-declaration and multi-origin feature. This is followed by a presentation of the system-wide discount and double-discount problem in Chapter 4. Later in Chapter 5, we illustrate the pivot-weight scheme in the airfreight consolidation. Finally, Chapter 6 concludes and highlights future research.

Chapter 2

Literature Review

In this chapter, we will walk through the literature in Sections 2.1 through 2.3 leading to our first research problem. Upon completion of the unsplittable shipment consolidation problem, literature in Sections 2.4 and 2.5 will inspire us to explore additional discount and pricing structures in airfreight, thus leading to the system-wide discount and pivot weight problem. Finally, the three research problems are largely supported by exploring the solution literature in Section 2.6.

2.1 Airfreight Consolidation Problem

The literature is rich with research on shipment consolidation problems. According to Hall (1987) and Higginson and Bookbinder (1994), there are three dimensions to freight consolidation: unit consolidation, time consolidation, and route consolidation. The first to formulate the airfreight-forwarder consolidation problem (AFCP) are Huang and Chi (2007). They consider a single origin with deterministic customer demand and formulate the problem as an MIP. They use Lagarangian relaxation to transform the model to a set covering problem, and develop a recursive heuristic that generates solutions that are often close to optimality. With similar assumptions and problem definition, Wong et al. (2009) formulate a forwarder's shipment planning problem using binary variables. Effects of integration and consolidation on the timely delivery of shipments during any phase of the shipping process are explicitly addressed. A forwarder's in-house capacity, as well as the available capacity of its partners and sub-contracting agents, are incorporated. The objective is to minimize the shipping cost subject to target delivery time, target cost, and resource capacity. Tabu search is used to solve the problem. Leung et al. (2009) consider a similar model and solve it using heuristics and branch-and-bound.

Compared to Huang and Chi (2007), neither Wong et al. (2009) nor Leung et al. (2009) exploit the difference between volumetric weight and gross weight for consolidation purposes. Moreover, neither take the bumping clause into account. Wong et al. (2009) and Leung et al. (2009) divide the forwarding process into k jobs, and try to get the best job-shipment assignment combination with minimal operational cost. Unlike Wong et al. (2009) and Leung et al. (2009), however, Huang and Chi (2007) consider the *readiness* of flights for shipment in terms of shipment ready time, the arrival due-time, or the requirements or preferences of shippers.

The consolidation of ocean freight is very similar to the airfreight problem. Ang et al. (2007) consider the *sea-cargo-mix problem* for the international shipment of ocean containers. It is similar to our problem, as there is consideration of shipment weight, volume and bumping clause. Those authors describe the characteristics of the cargo-mix problem for the carrier in a multi-period planning horizon, and formulate it as a multidimensional multiple knapsack problem (MDMKP). In particular, the MDMKP maximizes the total profit generated by all freight bookings accepted in a multi-period planning horizon, subject to the limited shipping capacities. Ang et al. (2007) propose two heuristic algorithms that can solve large scale problems, with tens of thousands of decision variables in a short time. They also conduct numerical experiments on randomly generated problem instances.

Table 2.1 compares the four papers that are most related to our research. Except for Wong et al. (2009), much of the earlier research concentrates on decisions for only a single origin. After various mergers and acquisitions, global freight forwarders are gaining more presence around the world. An airfreight forwarder thus needs to deal with decisions on multiple origins.

Paper	Model	Bumping Clause	Multi-Origin	Solution Methodology
Huang and Chi (2007)	MIP	Yes	No	Lagrangian Relaxation and Heuristic
Ang et al. (2007)	MCMKP	Yes	No	Heuristic
Leung et al. (2009)	IP	No	No	Branch-and-bound and Tabu Search
Wong et al. (2009)	IP	No	Yes	Tabu Search

Table 2.1: Comparison of Shipment Consolidation Literature Related to our Problem

2.2 The Multicommodity Flow Model for Consolidation of Splittable Shipments

Many of these previous studies on consolidation problems make use of a multicommodity flow model, where we can utilize the integral property and flow decomposition property of the commodity flow. Some related research is derived from the studies on courier delivery network design. Barnhart et al. (2002), Armacost et al. (2004), Barnhart and Shen (2005), Root and Cohn (2008), and Schenk and Klabjan (2008) incorporate network-level decision models. However, none of them take into account the bumping clause for consolidated shipments. Most of the models are based on the Piecewise-Linear Network Flow Problem (PLNFP), which concerns the minimization of a convex separable piecewise-linear objective function, subject to linear constraints. There are many successful solution methodologies for PLNFP, such as Balakrishnan and Graves (1989), Amiri and Pirkul (1997), Croxton et al. (2003) and Chang (2008). Balakrishnan and Graves (1989) formulate the PLNFP as a mixed integer program and develop a composite algorithm to generate both lower bounds and feasible solutions. However, those authors do not consider capacities nor over-declaration. Amiri and Pirkul (1997) use Lagrangian relaxation to solve this problem.

A common character of the consolidation problem modeled in PLNFP is that it assumes a demand between a pair of origin and destination nodes, and this demand is split across different route combinations. We use the term *splittable problem* to refer to the case of shipment consolidation that allows a load to be broken up and then reconsolidated. For simplicity, only the shipments gross weight is considered.

The splittable model has previously been formulated using three sets of MIP formulations: the *multiple choice model*, the *incremental model* and the *convex combination model*. Our models in Chapters 3, 4 and 5 are the "unsplittable" variants that are developed based upon the multiple choice model, first proposed by Balakrishnan and Graves (1989). Croxton et al. (2003) compare three mixed-integer programming formulations, each of which approximates the cost function by its lower convex envelope to solve a standard minimization problem with separable nonconvex piecewise linear costs.

The incremental model, presented initially by Dantzig (1960) and Hadley (1964), introduces a range-load variable y_j^k, defined as the load in the range k on arc j. (The value of this variables differs from that in the multiple choice model). Feasibility requires a value zero on range $k+1$ unless range k is full (reaches b_j^k). In other words, $w_j^k = 1$ if $y_j^k > 0$, but $w_j^k = 0$ otherwise. In addition to the manipulation of flow variable y, the new fixed cost is $\hat{F}_j^k = (F_j^k + c_j^k b_j^{k-1}) - (F_j^{k-1} + c_j^{k-1} b_j^{k-1})$. Therefore, the formulation appears as below:

$$\text{[Incremental Model]} \min \sum_j \sum_k (c_j^k y_j^k + \hat{F}_j^k w_j^k) \tag{2.1}$$

$$\text{s.t. } Ny = d \tag{2.2}$$

$$(b_j^k - b_j^{k-1}) w_j^{k+1} \leq y_j^k \leq (b_j^k - b_j^{k-1}) w_j^k \tag{2.3}$$

$$w_j^k \in \{0,1\}, \;\; y_j^k \geq 0$$

Note that in (2.2), the flow balance constraint is expressed in simplified rectangular-matrix form. In this formulation, $y_j^{K+1} = 0$ for the rightmost piecewise linear segment K of the cost function.

The convex combination model, suggested by Nemhauser and Wolsey (1988), does not handle discontinuous cost functions. Croxton et al. (2003) modify it such that the cost of flow within weight range k is a convex combination of the cost of its two endpoints b_j^{k-1} and b_j^k. By defining multipliers μ_j^k and λ_j^k as the weights on the two endpoints, the model can be re-written as:

$$\text{[Convex Combination Model]:} \min \sum_j \sum_k \mu_j^k (c_j^k b_j^{k-1} + F_j^k) + \lambda_j^k (c_j^k b_j^k + F_j^k) \tag{2.4}$$

$$\text{s.t. } Ny = d \tag{2.5}$$

$$\mu_j^k + \lambda_j^k = w_j^k \quad \forall j \tag{2.6}$$

$$\sum_k w_j^k \leq 1 \quad \forall j \tag{2.7}$$

$$\mu_j^k,\ \lambda_j^k \geq 0,\ w_j^k \in \{0,1\}$$

Given that all three of the previous formulations are valid, it is natural to ask if one is better than another. Croxton et al. (2003) establish that all three formulations are equivalent, in the sense that each approximates the real cost function with its lower convex envelope.

Chang (2008) also presents a heuristic solution to the courier delivery network design problem. However, none of the authors above honors the *integral property* of a shipment: It cannot be split along the path from origin to destination. The “unsplittable” requirement is central to all the three topics we worked on, and is discussed next.

2.3 Network Design for the Unsplittable-Shipments Consolidation Problem

One feature that distinguishes the design of an unsplittable shipment network from the standard network flow problem is that each commodity (shipment) must run through a single path in the network and cannot be separated. Bartolini and Mingozzi (2009) use the term *non-bifurcated.* If the flow is allowed to be divided among several paths, the problem is called the *bifurcated* or *splittable shipment problem.*

Belaidouni and Ben-Ameur (2007) refer to the unsplittable problem as the minimum cost single-path routing problem, commonly encountered in communication networks. When single-path routing is considered in a packet network, the sequence in which packets arrive is the same as their departure order, removing the need to implement any mechanism to support in-order packet delivery. Those authors present a cutting plane algorithm to solve the minimum cost multiple-source unsplittable flow problem.

Barnhart et al. (2000) solve an unsplittable flow problem arising in express package delivery. The majority of methods proposed to solve this problem are heuristics. For example, Crainic et al. (2000) and Ghamlouche et al. (2003) use a Tabu search approach. Atamtürk and Rajan (2002) study both the bifurcated and non-bifurcated problem, and report computational results on a set of instances corresponding to directed non-bifurcated problems. They present a linear time algorithm for separating residual capacity inequalities and introduce two new classes of valid inequalities for the non-bifurcated case, both of which include the "c-strong" inequalities as a special case. The effectiveness of the new inequalities is then tested using a branch-and-cut procedure. We note that none of the papers for the unsplittable problem considers economies of scale, nor the bumping clause in the unit transportation rates.

2.4 System-wide Discount and Double Discount

In traditional transportation problems, freight charges and discounts are associated with each transportation leg or route; there is no interaction between different legs or routes. In the trans-Atlantic market, we often encounter another type of discount called a *system-wide discount*: The carrier provides certain discounts if the forwarder can reach a specified aggregate capacity on the total flights operated by the carrier in a certain period of time. Thus, the forwarder's cost associated with one flight depends not only on the quantity of freight assigned to that flight, but also on the loads assigned to other flights operated by that carrier as well (Cohn et al., 2008). This type of discount scheme is typically offered by a regional airline whose focus is on the passenger business, but which offers its lower cabin for cargo. In that case, however, there are normally no quantity discounts on different weight ranges as in Figure 1.2(a). The reason is that the system-wide-discount scheme targets small-package transportation. Cohn et al. (2008) modeled it as a variation of the multi-commodity flow problem. Since almost nobody except those authors has explored this problem, we decided to further extend it with some practical features in Chapter 4.

2.5 Airfreight Consolidation with Pivot-Weight

Earlier works on airfreight consolidation problems assume that forwarders reserve cargo space from airlines in terms of payload or chargeable weight. This assumption is applied to general cargo and non-palletized shipments. However, large airfreight forwarders usually make their booking with an airline in terms of *ULDs*. Other than in Li et al. (2009), the pivot-weight scheme has not been investigated before.

In the meantime, quite a number of equivalent works on the ocean counterpart address the loading problem based on 20-foot containers. Pisinger (2002) investigates an ocean container-loading problem, that of loading a subset of rectangular boxes into a rectangular container of fixed dimensions, such that the volume of the packaged boxes is maximized. Brønmo et al. (2010) present a Dantzig-Wolfe procedure for the ship-scheduling problem with flexible cargo sizes. Although those discoveries on the ocean cargo side give some insight for us, ocean containers have a conventional fixed capacity limit, instead of a pivot-weight.

Our ACPW problem is different from the bin-packing and knapsack problems which have been used as the foundations of many consolidation-related problem variants. In bin-packing, cost is associated with the bin-level, but no charge is laid based on the weight or volume of items as in ACPW. The decision variables for bin packing are all binary.

Our problem, however, also has a continuous "overage" cost compared to the variable-size-and-cost bin-packing problem. In addition, we have distinct capacities on different bins. The open-end bin-packing problem (Leung et al., 2001) allows the capacity of each bin to be exceeded by only one of the items packed into the bin; whereas our ACPW does not restrict the number of excess items. Our problem also differs from the conventional knapsack problem: First, we have multiple knapsacks with diverse capacities. Second, we have an over-pivot cost per unit of cargo that exceeds the pivot weight.

There is some existing research on bin-packing problems which gives much insight to tackle our problem. In particular, the variable-size bin-packing problem (VSBPP) resembles our work without the continuous variable that represent the over-pivot weight. However, it should be noted that an important assumption was made in this literature with respect to the fixed bin costs, which are defined as being proportional to volumes of the corresponding bins. Although this assumption yields an easy approximation, it does not reflect reality in the transportation industry. In the airfreight business, the ULD reservation fee is a relatively independent attribute, one that may or may not be correlated to its capacity. Therefore we lift this hypothesis in VSBPP in our ACPW, by considering a fixed reservation cost independent of ULD capacity. Crainic et al. (2011) address the independence of the fixed reservation cost and the bin capacity, but do not account for the unit cost c_j and the over-pivot cost c_j^E.

2.6 Solution Methodology Literature

The core approach that we use to solve large instances of the three research problems is decomposition. We will review how other literature leverages on branch-and-price and column generation in this section. After that, we discuss a heuristic approach called local branching for large MIP problems.

2.6.1 Lagrangian Relaxation Techniques

Lagrangian relaxation works by dualizing a set of constraints that produces an easily solvable Lagrangian subproblem (SP) (Fisher, 1981). Lagrangian techniques have been used extensively to solve the capacitated multi-commodity network flow problem with piecewise linear concave costs (Muriel and Munshi, 2004). For minimization problems, the optimal value of the subproblem is a lower bound (LRLB) to the optimal value of the original problem (Martin, 1999). The Lagrangian dual problem can be formulated as a master problem (MP), which yields a Lagrangian upper bound (LRUB). The master problem, subproblem and the multipliers are updated iteratively as shown in Figure 2.1. When the Lagrangian upper and lower bound coincide, the solution is often infeasible to the original MIP. Therefore, a Lagrangian heuristic is used to construct a feasible solution. When this Lagrangian technique is embedded into branch-and-bound tree, the resulting approach is called branch-and-price.

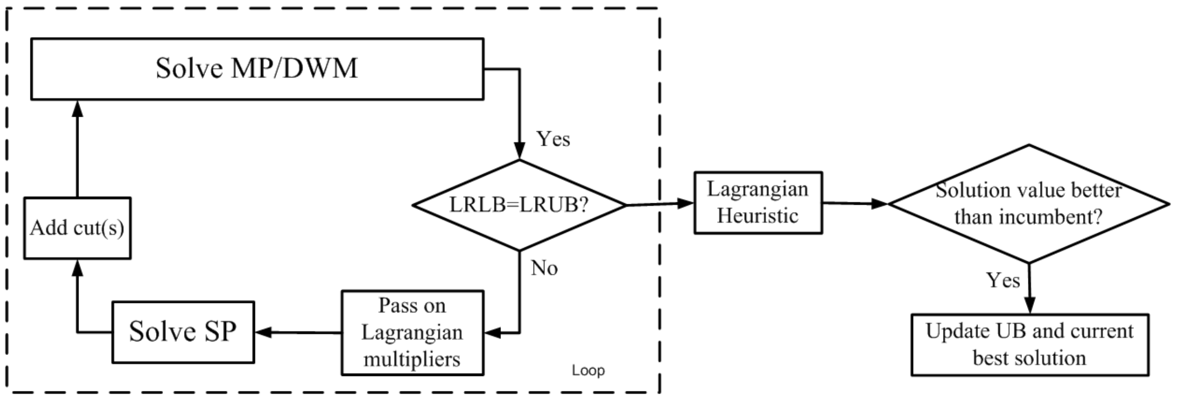

Solve MP/DWM
Yes
LRLB=LRUB?
Add cut(s)
No
Solve SP
Pass on Lagrangian multipliers
Loop
Lagrangian Heuristic
Solution value better than incumbent?
Yes
Update UB and current best solution

Figure 2.1: The Flow of Typical Lagrangian Relaxation Techniques

2.6.2 Branch-and-Price and Column Generation

To find efficient solution methodologies for the three topics we worked on, we studied the solution methods for the capacitated network design problem (CNDP) extensively. For a CNDP problem, since the gap between the optimal IP value and the optimal value of the LP-relaxation at the root node is usually very large, the LP-relaxation lower bounds are weak and the conventional branch-and-bound algorithm is not efficient (Puchinger et al., 2010). Lagrangian relaxations yield substantially tighter lower bounds than simpler LP relaxations in many MIPs (Frangioni, 2005). Upper bounds (for minimization problems) are often obtained as intermediate by-products from the subgradient procedure or by applying Lagrangian heuristics such as rounding or repair procedures. Even without embedding into a branch-and-bound framework, those Lagrangian-relaxation based methods can sometimes be turned into highly successful heuristic approaches.

In the meantime, column generation is at the heart of most of the exact solution approaches based upon Lagrangian relaxation. Column generation solves a model that results from the Dantzig-Wolfe decomposition of a compact formulation, is stronger, and leads to improved lower bounds (Vance, 1998; Alves and de Carvalho, 2007). Column generation has been used widely for the purpose of solving large scale mixed-integer problems. The original mixed integer problem is decomposed into a master problem and a subproblem using the Dantzig-Wolfe decomposition. The master problem contains a first subset of the columns. The subproblem is a separation problem for the dual, which is solved to identify whether the master problem should be enlarged with additional columns or not. The procedure iterates between the master problem and the subproblem, until the former contains all the columns that are necessary to span the optimal solution of the original MIP (Cherfi and Hifi, 2010). When column generation is performed for each node of the branch-and-bound tree, the approach becomes branch-and-price (Puchinger et al., 2010). One main advantage of column generation is to push certain complex constraints outside of the Dantzig-Wolfe master problem to an auxiliary problem (subproblem) (Chabrier, 2003). For the network design problem, the resulting decomposed model usually contains the "path column" that represents a feasible path, defined by the ordered list of the visited nodes. In Chapter 3, we have a chance to decompose the unsplittable shipment consolidation problem into a shortest-path/network-flow problem, which makes the overall problem simpler.

However, it is well known that column generation procedures suffer from slow convergence induced by undesirable behavior such as primal degeneracy, or the excessive oscillations of the dual variables. We can leverage on many existing techniques to speed up this process, such as in Alves and de Carvalho (2007) and Elhedhli et al. (2011). In recent years, much effort has been devoted to the topic of stabilized column generation, with the purpose of accelerating these processes. One way of accelerating column generation was proposed by Degraeve and Peeters (2003), who use a simplex method/subgradient-optimization procedure to solve the LP relaxation of the cutting stock problem. To obtain the optimal solution, the following procedure is repeated: for a specific number of iterations, subgradient optimization is used to update the dual prices, and new columns are priced and generated; then, the simplex method is used to reoptimize the master problem with the new columns added.

Column generation and Lagrangian relaxation are equivalent (Nemhauser and Wolsey, 1988), and the subgradient method has been extensively used to solve the Lagrangian problem. Column generation is known to be more robust, but it has the burden of reoptimizing the master problem to update the dual variables at every iteration. The subgradient method provides a fast way of updating the dual solution, but may have some convergence difficulties. The hybrid procedure of Degraeve and Peeters (2003) combines the robustness of the column generation method with the fast update of the dual prices of the subgradient method, producing an overall acceleration of the solution process. We are going to use subgradient optimization for our problems in Chapters 3 and 4.

For the pivot-weight scheme that we discuss in Chapter 5, most methodologies in the literature rely either on decomposition techniques (Alves and de Carvalho, 2007) or on reformulations solved using commercial MIP software, and require significant computation times when applied to large instances. This computational effort makes them difficult to use in practice, for planning settings in particular, where efficient, i.e., fast and accurate, solution methods are crucial to address the ACPW subproblems that must be solved repeatedly.

With regard to the slow convergence in column generation, a neighborhood or local-search heuristic presents an alternative approach to searching for solutions. These techniques sacrifice the guarantee of obtaining an optimal solution to find high quality solutions quickly. Instead of producing solutions by searching systematically defined regions of the feasible set, these heuristics iteratively define and search a neighborhood of a known solution for an improving solution. Thus, critical decisions when designing a local search heuristic are the size of the neighborhood to be searched at an iteration, and whether its structure enables the neighborhood to be searched efficiently. Meta-heuristics such as in Li et al. (2009) have been quite successful at avoiding becoming stuck at bad local optima. Although often successful at producing high-quality solutions, local search heuristics are typically unable to furnish a bound on the optimal value of the problem. Thus, the only measure of solution quality is a comparison with solutions obtained by alternative approaches.

2.6.3 Local Branching

Fischetti and Lodi (2003) introduce the local branching framework for MIP, tailored for encompassing the advantages of branch-and-bound and neighborhood search. This framework uses a general mixed-integer programming solver (CPLEX-MIP solver) to explore solution subspaces or neighborhoods defined by introducing linear inequalities in a mathematical model of the problem. During local branching, a k-OPT neighborhood of a feasible solution is searched, via a k-OPT neighborhood constraint, in the hopes of better nearby feasible solutions. Once the neighborhood is completely searched, the complement constraint is imposed on the general MIP. Local Branching works by partitioning the search tree through so-called local branching cuts. Since those local cuts are just specific constraints for integer programming problems, they can be expressed as normal IP constraints using any generic MIP solver.

Consider an MIP with 0-1 variables x, where $x = (x_1, ..., x_n)$ is the solution vector. A k-opt neighborhood around a given incumbent solution $\bar{x}$=$(\bar{x}_1, ..., \bar{x}_n)$ can be defined by the local branching constraint

$$\Delta\{x, \bar{x}\} := \sum_{j \in \bar{S}} x_j + \sum_{j \in SS \setminus \bar{S}} x_j \leq k \tag{2.8}$$

where $\bar{S}$ corresponds to those 0-1 variables that are set to one in the incumbent solution, i.e. $\bar{S} = \{j \in J | \bar{x_j} = 1\}$. $\Delta\{x, \bar{x}\}$ works as the conventional Hamming distance between integer solutions x and $\bar{x}$ (Puchinger et al., 2010). Starting from an initial solution, the search space is partitioned into the k-opt neighborhood of this incumbent (we often call it the "local tree") and the rest of search space by applying the local branching constraint (2.8) and its inverse $\Delta(x, \bar{x}) \geq k + 1$. The main phases of the standard local branching are as follows (Akeb et al., 2011):

1. Generate an initial solution for the first tree (node) - $\bar{x}$.

2. Initialize the first local tree (2.8), using the previously generated solution.

3. Iterative Step

   (3.1): Solve completely the local tree using an MIP solver.

   (3.2): When the local search terminates,

   - if a better feasible solution x' is found, then create from this local tree a new local tree ($\Delta(x', \bar{x'})$) using the improved solution x' as the initial solution;
   - otherwise, no better feasible solution is found; therefore, stop.

4. Solve the rest of the search tree corresponding to the inverse local branching constraint ($\Delta(x, \bar{x}) \geq k + 1$).

5. Return the best solution found.

Local branching combines mathematical programming with local search techniques such as intensification/diversification mechanisms. Compared to branch-and-price, local branching avoids solving a Danzig-Wolfe problem at each node of the branching tree. The power of Local branching in solving difficult MIP problems has been demonstrated in Fischetti et al. (2004) for a telecommunications network design problem. Those authors and Rodriguez-Martin and Salazar-Gonzalez (2010) state that the approach is particularly suited for MIP problems in which the collection of binary variables can be partitioned into two sets, such that fixing the value of variables in the first group produces an easier-to-solve subproblem.

However, if we directly apply the same framework as in Fischetti and Lodi (2003) to the three problems we worked on, the computational performance is rather poor.

Lichtenberger (2005) and Wallace (2010) addressed some deficiencies of the basic local branching framework implementation as follows:

- When more than one set of binary variables is involved, we are unable to express the importance of one set over another. For example, in our unsplittable network consolidation problem, when the assignment decision of shipments to flights is known, the weight-range decision is easily known.
- The local branching constraint possibly defines a very large neighborhood. Given a problem with n variables, a feasible solution has $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ neighbors within a Hamming distance of k (the local tree includes all neighbors with a Hamming distance not larger than k, so the actual search tree is even larger)
- When there is more than one initial feasible solution available for local branching, the original local branching framework supports the exploration of only a single neighborhood, despite the fact that an optimal solution may come from any of the promising solutions.
- While a local branching constraint defines a neighborhood around a feasible solution, it provides no further guidance for exploring this neighborhood besides the standard branch and cut strategies (e.g. best-bound-first search). Other local search heuristics might help to tighten the search tree.

In order to address the shortcomings of basic local branching, we have added three extensions to the standard local branching algorithm. The first caters to exploration of multiple levels of binary variables; it is applied to both the unsplittable network consolidation (Section 3.3.2) and the system-wide discount problem (Section 4.5.1). The second one eliminates the restriction of sequential execution by allowing the creation of new local trees before the previous one(s) are finished. The third extension tries to reduce subproblem complexity by fixing variables that are less likely than others to change in the optimal result. Our final two extensions are applied to the Pivot-Weight problem in Section 5.5.

Chapter 3

The Unsplittable Shipment Consolidation Problem

3.1 Problem Description

The transportation of an entire load, without splitting it, facilitates the tracking of that shipment by vendor and customer. Indeed, in many situations, shipment splitting is not possible. Even if it were allowed, the result could be managerially cumbersome and lead to a logistics manager's nightmare, as orders are received in multiple packages through different deliveries. Barnhart et al. (2000) and Belaidouni and Ben-Ameur (2007) do satisfy the unsplittable property in solving an express-package delivery problem and a communication-packet routing problem, respectively. However, neither paper considers economies of scale, nor the bumping clause in the unit transportation rates.

In this chapter, the unsplittable requirement is applied to given a set of shipments with different origins and destinations. Moreover, each shipment is characterized by its gross and volumetric weight. Each flight has its capacity, as well as weight ranges with its associated cost. Freight forwarders are able to over-declare their consolidated shipments to the next weight range on all flights. There is a also a fixed cost associated with each flight chosen to be used, while the unit charge on each flight is based on the chargeable weight of the consolidated load on that flight. The freight forwarder's objective is to minimize the total freight cost plus the total fixed charges.

3.2 Problem Formulation

Despite its focus on a single origin, the model of Huang and Chi (2007) is the most suitable one for the unsplittable shipment consolidation problem in Section 3.1. We will improve that model by allowing shipments to be transported via hubs and multiple flight legs. As the airline industry operates in a network, many consolidations are made in consideration of the whole airfreight forwarding network, instead of a single station point. In practice, a global freight forwarder makes shipment decisions for multiple origins. Therefore, it would be better to take all stations in the network into account.

Based on this model, we propose an extension through an arc-based formulation. The unsplittable problem is a constrained version of the linear multicommodity flow problem, in which the flow of a commodity (defined in this case by an origin-destination pair) may employ only a single path from origin to destination. We also take shipment volume into account in considering the chargeable weight. We use $i \in I$, $j \in J$, $k \in K$ and $p \in P$ to denote, respectively, shipments, flights, weight ranges and nodes in the network.

The parameters are:

- R_j^k: cost per unit weight of flow on flight j, if the total weight falls in weight range k. For each j, $R_j^1 \geq R_j^2 \geq ... \geq R_j^K \geq 0$.
- b_j^k: end point of chargeable weight range k for flight j. $b_j^0 = 0$, unless stated otherwise.
- g_i: gross weight of shipment i;
- v_i: volumetric weight of shipment i.
- J_p^-: set of flights whose origin is p;
- J_p^+: set of flights whose destination is p;
- N_i^-: origin node of shipment i;
- N_i^+: destination node of shipment i;
- T_{jp}^-: departure time for flight j at airport p;
- T_{jp}^+: arrival time for flight j at airport p;
- Q_i^-: ready time at origin for shipment i;
- Q_i^+: delivery deadline for shipment i at the destination;

The decision variables are:

- $w_j^k = \begin{cases} 1, & \text{if the chargeable weight of combined shipments} \\ & \text{falls in weight range } k \text{ of flight } j; \\ 0, & \text{otherwise;} \end{cases}$

- $z_j^i = \begin{cases} 1, & \text{if shipment } i \text{ is assigned to flight } j; \\ 0, & \text{otherwise;} \end{cases}$

- y_j^k is the chargeable weight on flight j (including all shipments consolidated on that flight) that falls in weight range k;

The mathematical formulation is:

Problem [P-UNSPLIT]

$$\min \sum_j \sum_k R_j^k y_j^k \tag{3.1}$$

$$\text{s.t.} \sum_{j \in J_p^+} z_j^i - \sum_{j \in J_p^-} z_j^i = \begin{cases} -1 \text{ if } p = N_i^- \\ 1 \text{ if } p = N_i^+ \\ 0 \text{ for all other nodes} \end{cases} \quad \forall p, i \tag{3.2}$$

$$\sum_i g_i z_j^i \leq \sum_k y_j^k \quad \forall j \tag{3.3}$$

$$\sum_i v_i z_j^i \leq \sum_k y_j^k \quad \forall j \tag{3.4}$$

$$b_j^{k-1} w_j^k \leq y_j^k \leq b_j^k w_j^k \quad \forall j, k \tag{3.5}$$

$$\sum_k w_j^k \leq 1 \quad \forall j \tag{3.6}$$

$$\sum_{j \in J_p^-} T_{jp}^- z_j^i \geq Q_i^- \quad \forall i \text{ and } p = N_i^- \tag{3.7}$$

$$\sum_{j \in J_p^+} T_{jp}^+ z_j^i \leq Q_i^+ \quad \forall i \text{ and } p = N_i^+ \tag{3.8}$$

$$\sum_{j \in J_p^-} T_{jp}^- z_j^i \geq \sum_{j \in J_p^+} T_{jp}^+ z_j^i \quad \forall p, i \tag{3.9}$$

$$w_j^k \in \{0, 1\}; y_j^k \geq 0 \tag{3.10}$$

$$z_j^i \in \{0, 1\} \tag{3.11}$$

We refer to this formulation as the *original problem* or *full-size problem* because it takes the shipment volume information into account. Constraint (3.2) ensures flow balance over the network. It requires that a shipment will first board a flight that starts at its origin and boards a flight that goes to its destination. For a hub node (that does not match a shipment's origin or destination), we need to choose a flight that flies to this node, and a flight that flies out of this node. Constraints (3.3) and (3.4) are imposed to make chargeable weight equal to the maximum of the gross and volumetric weights. Inequality (3.5) implies that if a particular weight range is selected ($w_j^k = 1$), then the flow y_j^k is bounded by the lower and upper weight breaks b_j^{k-1} and b_j^k respectively. Constraint (3.6) governs the selection of at most one weight range on a particular flight. Note that this problem is not decomposable by origin-destination pair, as the cost on each flight depends on the total weight consolidated on that flight.

Chang (2008) presents a shipment consolidation model whose objective is to minimize total travel time. However, the assumption that a shipment can board the very next flight, any time after those goods arrive at the airport, is not realistic. In practice, each shipment has a ready time Q_i^- and a delivery deadline Q_i^+. Each flight has a departure time T_{jp}^- at origin and an arrival time T_{jp}^+ at the destination. We have constraints (3.7) and (3.8) so that the flight carrying a particular shipment would leave the origin only after the shipment ready time and arrive before the delivery deadline. Constraint (3.9) ensures the precedence relationship between a shipment's time of arrival at a hub, and its departure from there.

3.3 Solution Methodology

In this section, we propose two solution methodologies for the problem, based on Lagrangian relaxation and local branching.

3.3.1 The Lagrangian Relaxation Approach

Similar to Balakrishnan and Graves (1989) and Amiri and Pirkul (1997), we apply Lagrangian relaxation to *Problem P-UNSPLIT* by relaxing constraints (3.3) and (3.4) with non-negative multipliers α_j and β_j. This leads to the following subproblem:

Problem [P-UNSPLIT1]

$$\begin{aligned}\min \; & \sum_j \sum_k R_j^k y_j^k + \sum_j \alpha_j (\sum_i g_i z_j^i - \sum_k y_j^k) + \sum_j \beta_j (\sum_i v_i z_j^i - \sum_k y_j^k) \\ \text{s.t.} \; & \text{constraints } (3.2), (3.5) - (3.11)\end{aligned}$$

This subproblem can in turn be decomposed into two sets of subproblems:

Problem [P-UNSPLIT2]

$$\min \sum_k (R_j^k y_j^k - \alpha_j y_j^k - \beta_j y_j^k) \quad \forall j$$
$$\text{s.t. constraints } (3.5), (3.6) \text{ and } (3.10)$$

Problem [P-UNSPLIT3]

$$\min \sum_j (\alpha_j g_i + \beta_j v_i) z_j^i \quad \forall i$$
$$\text{s.t. constraints } (3.2), (3.7) - (3.9) \text{ and } (3.11).$$

Each of the $|J|$ subproblems P-UNSPLIT2 can be solved as a knapsack problem using a greedy algorithm. And each of the $|I|$ subproblems P-UNSPLIT3 can be solved as a network flow problem using CPLEX.

Subgradient Procedure

We start with the standard subgradient optimization method to solve the Lagrangian dual problem. Let Z_{lag}^{α} be the sum of objective values of relaxed problems P-UNSPLIT2 and P-UNSPLIT3. We also define Z^* as the best feasible solution found so far in all iterations.

Similar to Fisher (1981), the multipliers α_j and β_j are adjusted as follows:

$$\alpha_j^+ = \alpha_j + t(\sum_i g_i z_j^i - \sum_k y_j^k) \quad \forall j \tag{3.12}$$

$$\beta_j^+ = \beta_j + t(\sum_i v_i z_j^i - \sum_k y_j^k) \quad \forall j \tag{3.13}$$

The step-size function is given by

$$t = \lambda \frac{Z^* - Z_{lag}^{\alpha}}{\sum_j [(\sum_i g_i z_j^i - \sum_k y_j^k)^2 + (\sum_i v_i z_j^i - \sum_k y_j^k)^2]}$$

We set the initial value of $\lambda_0 = 2$. If the solution Z_{lag}^{α} does not improve in 20 consecutive iterations, let $\lambda_{r+1} = \lambda_r/2$. Terminate the algorithm after 800 iterations unless an optimal solution is reached before that point, or after 80 consecutive iterations if the best lower bound improves by a total of less than 0.01%. Ideally, the lower bound and upper bound would meet together, where we call it the *Lagrangian bound*.

Construction Heuristic

In each iteration of the subgradient algorithm, the solution of problem P-UNSPLIT3 gives a route for any pair of communicating nodes. This solution is feasible if the total load shipped on flight j is smaller than or equal to the flight capacity. If this condition is violated for one or more arcs, we proceed to the next subgradient iteration and no feasible solution is generated in the current iteration. The details are as follows:

Step 1 Let $\bar{z}$ be the solution generated by Problem P-UNSPLIT3 in the current subgradient procedure.

Step 2 Let $E_j = \max\{\sum_i g_i \bar{z_j^i}, \sum_i v_i \bar{z_j^i}\}$ for each flight j. If $\exists\ j$ such that $E_j \geq b_j^{|K|}$, then constraint (3.5) is violated and no feasible solution is generated in the current iteration. Go to *Step 3*, otherwise go to *Step 4*.

Step 3 For such a flight $j = \tilde{j}$ that has an infeasible solution in Step 2

```
    **If** there is another flight $j'$ that connects the same flight origin and destination
as
      $\tilde{j}$ (that violates the capacity constraint),
  **then** (based on $\bar{z}$ from Step 1)
      Sort all those shipments that take flight $j$ by their chargeable weight.
      Load the shipments on flight $j'$ in ascending order of their chargeable weight.
   **Else** Adjust the multipliers according to equations (3.12) and (3.13).
```

Step 4 For each flight $j \in J$, let k' be the smallest weight range whose upper limit is larger than or equal to E_j.

```
**If**
 $E_j = 0$, then no range is selected
**Else**,  For each flight $j$, weight range $k$,
  **If** $R_j^{k'} E_j > R_j^{k'+1} b_j^{k'}$ and $k' < |K|$
    **If** $k = k' + 1$
    set $y_j^k = b_j^{k'}$ and $w_j^k = 1$
    **Else**
```

```
        set $y_j^k = 0$ and $w_j^k = 0$
    Else
        If $k = k'$
        set $y_j^k = E_j$ and $w_j^k = 1$
        Else
        set $y_j^k = 0$ and $w_j^k = 0$
```

Step 3 is computationally intensive. Some authors thus suggest going directly to adjusting the multipliers, instead of performing a "fixing" operation as in Step 3. We have found, however, that Step 3 leads to more rapid convergence in those networks where often there is more than a single flight that connects each pair of nodes. Note that step 3 is trying to find an alternative flight that matches the origin/destination of the over-capacitated *flight*, and not the origin/destination of the shipment.

Column Generation

Instead of the subgradient procedure, we can utilize the column generation approach. At each iteration, we find the Lagrange multipliers by solving the *Lagrangian Master Problem*:

$$\max \sum_{j \in J} \theta_j + \sum_{i \in I} \theta_i$$

$$\text{s.t. } \theta_j \le \sum_k (R_j^k - \alpha_j - \beta_j) y_j^{k,h_1} \quad \forall j \tag{3.14}$$

$$\theta_i \le \sum_j (\alpha_j g_i + \beta_j v_i) z_j^{i,h_2} \quad \forall i \tag{3.15}$$

Note that we use the superscripts h_1 and h_2 to distinguish the decision variables at different iterations. The corresponding dual - Dantzig-Wolf Master problem is:

$$\min \sum_{h_1} \sum_{j} \gamma_j \sum_{k} R_j^k y_j^{k,h_1} \tag{3.16}$$

$$\text{s.t.} \sum_{h_2} \gamma_i \sum_{i} g_i z_j^{i,h_2} - \sum_{h_1} \gamma_j \sum_{k} y_j^{k,h_1} \leq 0 \quad \forall j \tag{3.17}$$

$$\sum_{h_2} \gamma_i \sum_{i} v_i z_j^{i,h_2} - \sum_{h_1} \gamma_j \sum_{k} y_j^{k,h_1} \leq 0 \quad \forall j \tag{3.18}$$

$$\sum_{h_1} \gamma_j^{h_1} = 1 \tag{3.19}$$

$$\sum_{h_2} \gamma_i^{h_2} = 1 \tag{3.20}$$

$$\gamma_j^{h_1}, \gamma_i^{h_2} \in \{0, 1\} \tag{3.21}$$

The relaxed subproblem provides a Lagrangian *lower bound*, and the Dantzig-Wolf Master problem provides an *upper bound* to the Lagrangian dual problem. The iterations are continued until the lower bound meets the upper bound, which equals the Lagrangian bound in the subgradient procedure.

3.3.2 Modified Local Branching Heuristic

As we mentioned in Section 2.6.3, the original implementation of Fischetti and Lodi (2003) treats all binary variables equally. In our problem formulation, we have two sets of binary variables z_j^i and w_j^k. However, when the z_j^i are fixed, the w_j^k become known. Therefore, we consider only z_j^i in the neighborhood exploration. For a given feasible solution $\bar{z}$ to problem P-UNSPLIT in Section 3.2, we define the *k-OPT neighborhood* $N(\bar{z}, \delta)$ of $\bar{z}$ as the set of feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(z, \bar{z}) : \sum_i \sum_{j \text{ and } z_j^i = 1} (1 - z_j^i) + \sum_i \sum_{j \text{ and } z_j^i = 0} z_j^i \leq \delta \tag{3.22}$$

Therefore, for a given incumbent solution $\bar{z}$, the solution space can be partitioned into a left branch and a right branch according to (3.23):

$$\Delta(z, \bar{z}) \leq \delta \text{ (left branch) }, \Delta(z, \bar{z}) \geq \delta + 1 \text{ (right branch)} \tag{3.23}$$

The above definition is consistent with the classical k-OPT neighborhood for the Traveling Salesman Problem. In the approach of Fischetti and Lodi (2003), neighbors around the incumbent solution are defined by adding constraints to the original model. The neighborhood-size parameter δ should be chosen as the largest possible value, such that the left-branch subproblem is much easier to solve than the one associated with the parent node. The idea is that the neighborhood $N(\bar{z}, \delta)$ corresponding to the left branch must be "sufficiently small" to be optimized in a short computing time, but still "large enough" to likely contain better solutions than $\bar{z}$. According to computational experience, the choice of δ in the range [11,20] is effective in most cases.

Our implementation follows the framework proposed by Fischetti and Lodi (2003), except that we consider only the first-level binary variable z_j^i in the initial neighborhood exploration. After adding constraint (3.22), if the system is not solved to proven optimality within a given time limit, we resort to the variation of second-level variables which are constrained by (3.24):

$$\Delta(w, \bar{w}) : \sum_{j} \sum_{k \ and \ w_j^k=1} (1 - w_j^k) + \sum_{j} \sum_{k \ and \ w_j^k=0} w_j^k \leq \delta_2 \tag{3.24}$$

By iteratively increasing the value of δ_2 up to δ_2^{max}, we explore the second-level neighborhoods which are contained in the first-level neighborhood defined by constraint (3.22). This modification, which caters to two levels of binary variables, yields excellent results for our test cases, compared to ordinary local branching as in Fischetti and Lodi (2003). (See Table 3.9.) The notations used in the algorithm are:

UB - the current problem upper bound. It is used to interrupt the optimization as soon as the best lower bound becomes greater or equal to UB;

$first\_feasible$ - boolean parameter for CPLEX MIP Solver which is $true$ if the first solution lower than UB is required as output; if $first\_feasible = false$, CPLEX returns the best solution found so far.

t_{max} - the overall running time limit, which is also the global stopping threshold;

z_{opt} - incumbent solution;

f_{opt} - the corresponding objective function value of the incumbent solution;

$node_{max}$ - the time limit for each tactical branching exploration;

dv_{max} - the maximum number of diversifications allowed (We use the value 10 in our algorithm);

elapsedtime - the current elapsed solution time;

$z_{cur}, w_{cur}, y_{cur}, f_{cur}$ - the current solution for variables and corresponding value of the objective;

The detailed implementation is illustrated in Figure 3.1. The algorithm starts with a feasible solution $\bar{z}_1$. The branching constraint $\Delta(z, \bar{z}_1) \leq \delta$ is added to the model creating a left-branch subproblem (node 2) that is solved by CPLEX. There are two cases we need to consider: (a) If a better solution $\bar{z}_2$ is found within the node time-limit $node_{max}$, then it becomes the new incumbent. The process backtracks to the father node (node 1), and the constraint $\Delta(z, \bar{z}_1) \leq \delta$ is replaced by $\Delta(z, \bar{z}_1) \geq \delta + 1$. A new left-branch node (node 4) is created by adding the cut $\Delta(z, \bar{z}_2) \leq \delta$ (without modifying the value of parameter δ). The above scenario is illustrated in Figure 3.2(a). (b) If the solution $\bar{z}_1$ is not improved within the node time limit, we reduce the size of the neighborhood in an attempt to speed-up its exploration. This can be achieved by reducing the right-hand side term by $\lfloor \delta/2 \rfloor$. Hence, node 2 is created as illustrated in Figure 3.2(b). This step resembles the *intensification* step in Tabu search. Whenever the current neighborhood exploration finds a new solution, δ is reset to its initially-chosen value, and exploration continues in the same fashion.

At the same time, a *diversification* step is applied in the local branching procedure. That is triggered when the MIP solver reports proven infeasibility, or when it is unable to find a feasible solution within the node time-limit. There are two diversification measures applied in our implementation. On one hand, we can enlarge the neighborhood of the reference solution $\bar{z}$, by increasing the current δ value. On the other hand, we can remove the upper bound on the optimal solution value, and add a constraint $\bar{z} \geq 1$ in order to escape from the current solution. We also set an upper limit of total diversification steps being applied. The termination criterion is either that the total time limit (3 hours) is reached, or the maximum number of diversification steps is exceeded.

3.4 Numerical Experiments

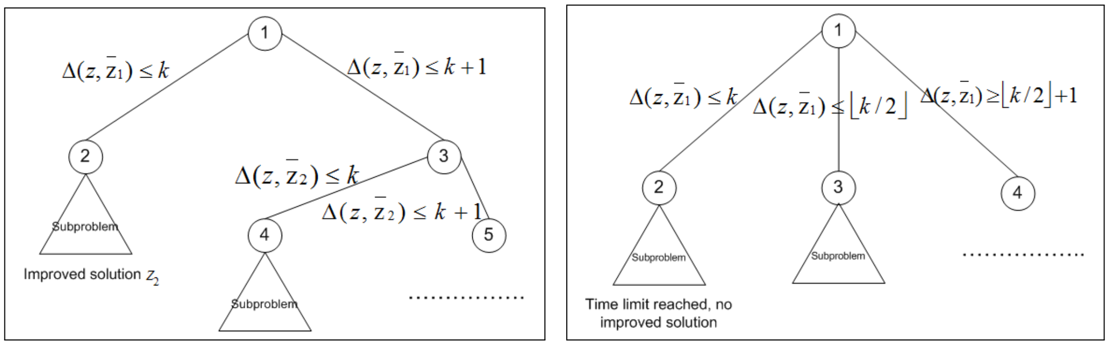
3.4.1 Generation of Test Cases for Numerical Analysis

In order to evaluate the effectiveness of the Lagrangian relaxation approach, we generated a test bed which consists of 18 cases in Table 3.1. Each case is characterized by four parameters: $|I|$, $|J|$, $|K|$, $|P|$. The nodes are picked randomly. First, the origin nodes are chosen, then the exporting hubs, the importing hubs, and finally the destination nodes. Amiri and Pirkul (1997) and Cohn et al. (2008) employ the parameter *arc density* to indicate the probability of having a direct flight between any node pair. As the network we generated is *layered*, we use the *network availability* parameter (Nav), which equals the average number of destinations (in the destination layer) that can be reached from a particular origin. (The generated network is fully connected unless otherwise mentioned.) The origin-destination pair for each shipment is randomly selected from a uniform distribution. Each case is generated with *flight capacity* of 80%, where the flight capacity (Fca) is:

$$Fca = \frac{\text{Total Chargeable Weight}}{\text{Total Flight Booking Capacity on flights between exporting and importing hubs}} \tag{3.25}$$

```
FUNCTION LOCALBRANCHING($\delta, t_{max}, UB, dv_{max}$)
 1  $UB = +\infty; first\_feasible = TRUE; TL = t_{max}; rhs = \delta;$
 2  while $elapsedtime < t_{max}$ or $dv > dv_{max}$
 3  do $TL = min(node_{max}, t_{max} - elapsedtime);$
 4       add local branching constraint $\Delta(z, z_{cur}) \leq rhs; UB = f_{cur};$
 5     $status = CPLEXSOLVE(TL, UB, first);$
 6     switch $status$
 7       case "$opt\_sol\_found$" :
 8               reverse last local branching constraint to $\Delta(z, z_{cur}) \geq rhs + 1;$
 9               $< z_{cur}, w_{cur}, y_{cur} >=< \hat{z}, \hat{w}, \hat{y} >; f_{cur} = \hat{f}; rhs = \delta; UB = f_{cur};$
10       case "$feasible\_sol\_found$" :
11               if $first\_infeasible$
12                 then remove last local branching constraint $\Delta(z, z_{cur}) \leq rhs;$
13                      $< z_{cur}, w_{cur}, y_{cur} >=< \hat{z}, \hat{w}, \hat{y} >; f_{cur} = \hat{f}; rhs = \delta; UB = f_{cur};$
14       case "$proven\_infeasible$" :
15               reverse last local branching constraint to $\Delta(z, z_{cur}) \geq rhs + 1;$
16               if $diversify$
17                 then $UB = TL = +\infty; dv++; first\_feasible = TRUE;$
18               $rhs = rhs + \lceil \delta/2 \rceil; diversify = TRUE;$
19       case "$no\_feasible\_sol\_found$" :
20               REFINING_HEURISTIC($\delta_2^{max}$)
21               if $diversify$
22                 then replace last local branching constraint to $\Delta(z, z_{cur}) \geq 1;$
23                      $UB = TL = +\infty; dv++; first\_feasible = TRUE;$
24                      $rhs = rhs + \lceil \delta/2 \rceil; first\_feasible = TRUE;$
25                 else remove last local branching constraint $\Delta(z, z_{cur}) \leq rhs;$
26                      $rhs = rhs - \lceil \delta/2 \rceil;$
27               $diversify = TRUE;$
28     endswitch
29     if $f_{cur} < f_{opt}$
30       then $< z_{opt} = z_{cur}, y_{opt} = y_{cur}, w_{opt} = w_{cur} >; f_{opt} = f_{cur};$
31  end while
32  $TL = t_{max} - elapsedtime; first\_feasible = FALSE;$
33  $status = CPLEXSOLVE(TL, first\_feasible)$
34  if $status =$ "$proven\_infeasible$" or "$no\_feasible$"
35    then return $f_{cur};$
36    else return $NOT\_FEASIBLE;$
```

Figure 3.1: Implementation of Local Branching


1
$\Delta(z,\bar{z}_1) \le k$
$\Delta(z,\bar{z}_1) \le k+1$
2
3
$\Delta(z,\bar{z}_2) \le k$
$\Delta(z,\bar{z}_2) \le k+1$
4
5
Subproblem
Improved solution z_2
Subproblem
1
$\Delta(z,\bar{z}_1) \le k$
$\Delta(z,\bar{z}_1) \le \lfloor k/2 \rfloor$
$\Delta(z,\bar{z}_1) \ge \lfloor k/2 \rfloor + 1$
2
3
4
Subproblem
Subproblem
Time limit reached, no improved solution

(a) Case (a) (b) Case (b)

Figure 3.2: Cases for Local Branching

There are four weight ranges on each flight ($|K| = 4$) with average *segment price variability* (Spv) of 2. *Spv* measures the mean difference in unit cost between adjacent weight ranges:

$$Spv = \frac{1}{K-1}\sum_{k=1}^{K-1}\frac{R_j^k}{R_j^{k+1}} \tag{3.26}$$

The lower the *Spv* ratio, the larger the cost difference between adjacent weight ranges. Lower *Spv* cases will have greater differences in slope values in Figure 1.2. The cost of the four weight ranges is generated uniformly over the interval $[1, 8]$, with decreasing segment cost from the first to the last segment on each flight.

As shipment volume information is considered here, we define *shipment density* (*Den*) as $\frac{\sum \text{Total Gross Weight}}{\sum \text{Total Volumetric Weight}}$. If *density* < 1, we call it a volume cargo case; otherwise, we call it a dense cargo case. All the test cases generated in this section will have average *Den* value of 1, unless otherwise mentioned.

Case	$\|I\|$	$\|J\|$	$\|K\|$	$\|P\|$
1	10	16	4	8
2	10	12	4	8
3	20	30	4	20
4	20	60	4	20
5	10	20	3	10
6	10	20	4	10
7	10	20	5	10
8	60	100	3	40
9	60	100	4	40
10-3	90	175	3	40
10-4	90	175	4	40
10-5	90	175	5	40
11-3	120	175	3	40
11-4	120	175	4	40
11-5	120	175	5	40
12-3	200	175	3	40
12-4	200	175	4	40
12-5	200	175	5	40

Table 3.1: Test Cases for Computational Experiments

Test cases 1 to 9 are smaller-size cases, while cases 10-3 to 12-5 reflect the business reality for a large freight forwarder's Sino-US tradelane. The number of weight ranges for Cases 10-12 is distinguished by the digits after the case number.

In addition to observing the effectiveness of the proposed solution methodologies, we analyze the impact on our problem difficulty of different parameter settings such as the number of weight ranges, weight range variability, number of shipments, and shipment weights.

3.4.2 Computational Analysis for Small-size Instances

We first tried to solve problem P-UNSPLIT using CPLEX 11.0 on a Sun Solaris machine with 1.6GHz CPU and 2GB of RAM. The goal was to investigate factors that affect the difficulty of the problem through some small-size instances, and seek insights for solving larger instances.

Initially, we did not consider weight ranges nor shipment volume, and imposed no capacity constraints in the network. So each shipment followed its cheapest path from origin to destination. We randomly generated 8 instances, each of which had shipment weight uniformly distributed over the interval [0,50], far below the first break point, 300. Each instance was solved within a few seconds at the root node. For multiple weight ranges, however, CPLEX required further branching with loads shifting from the higher cost range to ranges with some discounts. Moreover, with other parameters unchanged, when additional weight ranges were present, the solution became more challenging for tightly capacitated instances, but not for loosely capacitated instances. We also observed that the inclusion of fixed costs increases the problem difficulty and requires extra solution time.

All the factors mentioned above, however, seem to have less impact on the problem's ease of solution than the flight capacity or the number of shipments. We first varied only the flight capacity parameter, and generated test cases from base cases 2-4 in Table 3.1 as indicated in Tables 3.2 and 3.3 (The use of a letter suffix, as in Cases 2A or 2B, means that the network is that of Case 2, but there are slight differences in the value of Fca or Den).

Case	Nav	Fca	$\|I\|$	CPU Time (sec)
2A	12	0.2	10	1.0
2B	12	0.5	10	1.0
2C	12	0.8	10	14.2
3A	12	0.2	20	2.3
3B	16	0.5	20	7.6
3C	16	0.8	20	123.9
4A	24	0.2	30	2.9
4B	24	0.5	30	71.1
4C	24	0.8	30	466.4

Table 3.2: Impact of Flight Capacities when $|K| = 4$ and $Den = 1$

Instances with $Fca < 0.6$ are considered “loosely capacitated”, otherwise they are “tightly capacitated”. As shown in Table 3.2, Fca is the factor that most affects solution time. Instances whose Fca= 0.8 take about ten times as long to solve as those instances whose Fca= 0.5.

Some effects are not easy to observe when the number of shipments is small, so we generated larger instances A1 to D2. These cases have 60 to 400 shipments; all are based on a network with 40 nodes and 175 available flights. Along with flight capacities, the number of shipments is found to affect performance (Table 3.3). For networks of a given size, when the number of shipments increases, the computational performance of CPLEX deteriorates significantly. For an airfreight forwarder that handles 80 to 200 shipments per day, a solution time of more than 3 hours is definitely not acceptable. Therefore, it is necessary to find more efficient solution methodologies, which we now do by Lagrangian relaxation.

Case	$\lvert P\rvert$	$\lvert I\rvert$	Den	CPU Time(sec)
2	20	10	1	14.2
3	32	20	1	123.9
4	40	30	1	466.4
9R1A	40	60	0.8	506
9R1B	40	60	1.2	489
9R2A	40	120	0.8	4010
9R2B	40	120	1.2	4203
9R4A	40	200	0.8	8201
9R3B	40	200	1.2	$>$ 3hrs
9R4A	40	400	0.8	$>$ 3hrs
9R4B	40	400	1.2	$>$ 3hrs

Table 3.3: Impact of Number of Shipments when $Fca = 0.8$ and $|K| = 4$

3.4.3 Computational Analysis for Lagrangian Relaxation

In this section, we generate test cases to compare performance between the CPLEX MIP Solver and Lagrangian relaxation. The volumetric weight is randomly generated in the same way as the gross weight, with average shipment density equal to 1. In Table 3.4, we display the solution times for CPLEX, subgradient optimization and column generation side-by-side. In addition, “gap” measures the percentage difference between solution results and the known optimal solution of CPLEX. The “gap1” refers to the difference between the solution and the Lagrangian bound. Table 3.4 shows that the column generation approach consistently yields a wider gap from the optimal solution, although it provides substantial speedup (on average 25.6% faster) compared to subgradient optimization. This is because the solution space for column generation is smaller than that for subgradient optimization. Therefore, we use the results from subgradient optimization to represent Lagrangian relaxation from here onwards.

	CPLEX		Subgradient Optimization			Column Generation		
Test Case	%gap1	CPU Time (s)	% gap	% gap1	CPU Time (s)	% gap	% gap1	CPU Time (s)
1	21.1	9	0	21.1	125	0	21.1	40
2	24.2	8	0	24.2	101	0	24.2	43
3	28.0	289	0.2	28.3	1104	0.2	28.3	698
4	21.2	306	0.3	21.6	1360	1.5	23.0	723
5	9.0	40	1.3	10.4	198	1.2	10.2	191
6	15.8	10	0	15.8	205	3.0	19.3	183
7	19.0	22	0	19.0	209	0	19.0	194
8	15.7	362	1.7	17.7	3521	10.1	27.5	2681
9	22.3	518	1.4	23.9	4281	10.7	35.4	3980
9A	18.0	245	2.6	21.1	1164	11.7	31.7	825
9B	12.8	40	0	12.8	634	9.4	23.4	527
10-3	30.5	7508	1.3	31.7	2710	6.1	38.4	2302
10-4	27.6	7294	2.9	31.3	2586	4.9	33.8	2213
10-5	28.3	8054	2.7	31.8	2760	11.3	42.9	1899
11-3	8.8	5530	1.0	9.9	2820	9.1	18.7	2401
11-4	7.5	5620	3.0	10.7	1951	6.2	14.2	1542
11-5	8.5	6004	2.4	11.1	2654	11.1	20.5	2187
12-3	11.0	9891	1.4	12.5	2911	13.4	25.9	1640
12-4	6.4	10027	2.7	9.3	3033	5.3	12.0	2516
12-5	13.4	9650	1.7	15.4	3429	14.5	29.9	2998

Table 3.4: Comparison of Test Results between CPLEX and Lagrangian Relaxation

With the impact of volumetric weight and the requirement that each shipment should flow on a single path, the optimal value is higher than for the splittable case: Those consolidated shipments (commodities) cannot take advantage of some lower-cost paths. Because an increase in the number of weight ranges affects only the step of constructing a feasible solution in Lagrangian relaxation, the solution time is not very sensitive to that increase.

CPLEX is faster on the small-size cases 1 to 9, while Lagrangian relaxation outperforms CPLEX on the large problem series of 10 through 12. Cases 10-3 to 10-5 are "skewed cases". There are 10 origins, but all 90 shipments are sent from only two of them, in a network that is fully connected with 10 transshipment hubs and 20 destinations. Lagrangian relaxation takes about half the computation time of CPLEX for large problem instances e.g. Case 10-3.

Additional shipments in an extensive network do not necessarily imply longer computational times. Solution of Case 9A takes only half as long as Case 9. The average flight capacity is 85% in Case 9 and 70% for Case 9A. Case 9B further reduces the average flight capacity parameter to 35%, which is enough to carry all shipments on the least-cost flight, and requires a computation time of only 40 seconds. Cases 9A and 9B once again show that average flight capacity contributes to the difficulty of the problem.

3.4.4 Computational Analysis for Local Branching Heuristic

Based on Cases 10 and 12 in Table 3.1, we generated supplementary cases for computational analysis, respectively "loosely" and "tightly" capacitated in Tables 3.5 and 3.6. Each instance is characterized by the number of shipments $|I|$, number of flights $|J|$, number of weight ranges $|K|$ and number of nodes $|P|$. (Observe that $|P|$ equals the sum of the number of origins (L1), plus the number of exporting hubs (L2), the number of importing hubs (L3) and the number of destination nodes (L4).) The average volume and gross weights remain at 200kg. The first two digits of any case number denote the case in Table 3.1 from which it is derived. The next letter represents whether it is a “loose” or “tight” case. The last letter indicates capacity ratio, with A being the case having the highest Fca value. $Nav = 1$ or 2 in Tables 3.5 and 3.6 means the network is fully connected or partially connected, respectively.

Case	$\|I\|$	$\|J\|$	Fca	Nav
10L1	90	175	0.3	1
10L2	90	175	0.35	1
10L3	90	105	0.3	2
10L4	90	105	0.35	2
12L1	200	175	0.20	1
12L2	200	175	0.225	1
12L3	200	175	0.30	1
12L4	200	175	0.325	1
12L5	200	175	0.35	1

Table 3.5: Loosely Capacitated Test Cases with $|K| = 3$, $|P| = 40$, L1=10, L2=5, L3=5 and L4=20

By comparing the computational times in Tables 3.7 and 3.8, we can see clearly that capacity plays a vital role in the performance of the CPLEX MIP solver. The capacity has less impact for Lagrangian relaxation than for local branching. As the backbone solver for local branching is CPLEX, capacity obviously affects local branching. The most time-consuming part in the Lagrangian approach, the heuristic, is much less affected by the tightness of the capacity in each case. For tightly-capacitated cases, the average computation time for local branching is 49% of CPLEX, while time for Lagrangian relaxation is 41% that of CPLEX. We set a time limit for CPLEX MIP solver to be 3

Case	$\lvert I\rvert$	$\lvert J\rvert$	$\lvert K\rvert$	n. var	n. cons	Fca	Nav
10T3A	90	175	3	5,175	16,800	0.9	1
10T3B	90	175	3	5,175	16,800	0.8	1
10T3C	90	105	3	4,545	10,080	0.9	2
10T3D	90	105	3	4,545	10,080	0.8	2
12T3A	200	175	3	9,575	36,050	0.9	1
12T3B	200	175	3	9,575	36,050	0.875	1
12T3C	200	175	3	9,575	36,050	0.85	1
12T3D	200	175	3	9,575	36,050	0.825	1
12T3E	200	175	3	9,575	36,050	0.8	1
12T4A	200	175	4	9,925	36,400	0.9	1
12T4B	200	175	4	9,925	36,400	0.875	1
12T4C	200	175	4	9,925	36,400	0.85	1
12T4D	200	175	4	9,925	36,400	0.825	1
12T4E	200	175	4	9,925	36,400	0.8	1
12T5A	200	175	5	10,275	36,750	0.9	1
12T5B	200	175	5	10,275	36,750	0.875	1
12T5C	200	175	5	10,275	36,750	0.85	1
12T5D	200	175	5	10,275	36,750	0.825	1
12T5E	200	175	5	10,275	36,750	0.8	1

Table 3.6: Tightly-Capacitated Test Cases with $|P| = 40$, L1=10, L2=5, L3=5 and L4=20

hours for tightly-capacitated cases, and picked the best solution when that time limit was reached. (The characters 'TL' in the column *CPU Time* denote that attainment of the total time limit was reached.)

For solution quality, the percentage gap information is computed as $\frac{f_{method}-f_{cplex}}{f_{cplex}} \times 100\%$. The local branching approach ties or outperforms Lagrangian relaxation in 17 out of 18 cases. Moreover, the solution from local branching is on average only 0.8% from the optimal for tightly-capacitated cases, and 0.1% for loosely-capacitated cases. For Lagrangian relaxation, the average gap is 3.5% and 1.6% for these two categories

Case	CPLEX	Lagrangian Relaxation		Local Branching ($11 \leq \delta \leq 20$)	
	CPU Time(s)	%gap	CPU Time (s)	%gap	CPU Time (s)
10L1	108	1.6	1676	0	189
10L2	159	2.7	2081	0	231
10L3	12	0	1158	0	52
10L4	13	2.0	1402	0	66
12L1	22	2.2	1787	0	508
12L2	25	2.4	1406	0	497
12L3	184	1.2	2556	0	329
12L4	209	1.2	2534	0	281
12L5	211	1.2	2601	0.8	220

Table 3.7: Results for Loosely-Capacitated Cases

respectively. Local branching also outperforms CPLEX in three cases (12T5A, 12T5B, 12T5C) when the 3-hour time limit is imposed.

From Table 3.8, we find that for similar cases, higher flight capacity ratios (Fca) make the problem more difficult to solve. This trend is consistent for the three solution approaches. When the flight capacity is fixed, a larger number of weight ranges will slightly increase the solution times for CPLEX and the local branching approaches. However, the impact of the number of weight ranges is not so significant for Lagrangian relaxation.

Moreover, we can manipulate control parameters to force local branching to produce a feasible solution at earlier stages. Increasing the dv_{max} value, reducing $node_{max}$ and squeezing the neighborhood size δ can possibly serve this purpose. For example, by halving the current $node_{max}$ and doubling the value of dv_{max}, we reduced the gap from 10.7% to 5.4% for case $12T3A$. Table 3.9 implies that local branching provides early feasible solutions of good quality. When comparing results of the local branching heuristic to CPLEX after

Case	CPLEX	Lagrangian Relaxation		Local Branching ($11 \leq \delta \leq 20$)	
	CPU Time(s)	%gap	CPU Time(s)	%gap	CPU Time(s)
10T3A	4091	2.6	2561	1.2	1699
10T3B	2824	1.8	2698	1.8	1248
10T3C	1598	0.4	1909	0	907
10T3D	1930	0.9	1984	0	1054
12T3A	TL	7.9	4097	3.7	3688
12T3B	TL	1.6	4306	1.0	3601
12T3C	7510	2.3	4320	2.8	2360
12T3D	5833	2.2	3617	1.8	2524
12T3E	4206	1.0	3809	1.0	1833
12T4A	TL	12.7	4587	1.1	4608
12T4B	TL	2.6	4031	0.1	4287
12T4C	TL	2.1	3733	1.2	4275
12T4D	6997	2.4	3860	2.4	2965
12T4E	4101	2.4	3202	1.6	1070
12T5A	TL	10.2	4320	-1.0	6011
12T5B	TL	2.4	4159	-4.4	4830
12T5C	TL	1.2	3967	-2.4	4286
12T5D	7208	2.7	2983	1.4	3020
12T5E	4304	2.0	3146	0.8	2013

Table 3.8: Results for Tightly-Capacitated Cases

15 minutes of CPU time, local branching gives a better solution time for 10 out of 17 cases, and the same solution speed for another 2 instances (Table 3.9). If we compare the results from CPLEX after *an hour* of computational time with those given by the local branching in 15 minutes, we find that local branching outperforms CPLEX in three cases and ties in one.

In addition, Table 3.9 shows that using two levels of binary variables in local branching leads to faster convergence than considering all binary variables at a single level. For cases

Case	Our Local Branching (15 min)	Fischetti and Lodi (2003) (15 min)	Our Local Branching (60 min)	CPLEX (15 min)	CPLEX (60 min)
	%gap	%gap	%gap	%gap	%gap
12T3A	5.4	15.9	6.5	15.7	11.5
12T3B	4.9	5.9	2.7	4.9	3.5
12T3C	5.5	7.5	2.7	5.4	4.2
12T3D	5.3	6.3	1.9	3.4	2.9
12T3E	5.4	5.8	1.0	5.2	2.2
12T4A	3.2	7.5	1.8	6.7	3.2
12T4B	1.0	6.7	0.2	5.2	1.0
12T4C	2.8	6.3	1.4	2.8	1.1
12T4D	3.6	6.6	2.4	3.4	1.0
12T4E	2.9	4.2	1.6	2.9	0.8
12T5A	2.1	18.1	1.4	6.5	2.2
12T5B	3.8	3.7	3.2	3.8	1.7
12T5C	3.0	3.0	1.6	2.1	1.0
12T5D	1.9	3.8	1.4	2.5	1.6
12T5E	2.0	2.7	0.8	2.1	0.6
13T4A	0	7.9	0	11.2	0
14T5A	0	15.9	0	8.6	0

Table 3.9: Results for Tightly-Capacitated Cases with Different Time Limits

13T4A and 14T5A, our local branching method can even find optimal solutions within 15 minutes of time. Moreover, all those early solutions found by local branching are within 5.5% (3.5% on average) from the optimal solution computed by CPLEX after three hours. Comparing local branching and CPLEX with a 60-minute time limit, we find that local branching is faster than CPLEX in nine cases.

3.5 Summary

In this chapter, we provided a formulation for the airfreight forwarding problem that includes factors previously ignored in the literature, such as the volumetric weight, the bumping clause, the unsplittable requirement, and the flight/shipment times. After initial testing using a general MIP solver, we proposed two solution methodologies that are capable of handling large sizes and difficult instances in reasonable computational time. Heuristics based on Lagrangian relaxation and local branching approaches were explored and tested under different network structures.

In the remainder of this thesis, we investigate other cost structures that freight forwarders face. In particular, pricing based on *pivot weight* (the carrier will charge for this weight, even if the shipper's load is smaller), and *system-wide discounts* based on volume during a period, will be explored. The former is the subject of Chapter 5. Before that, we turn to the case of system-wide discounts.

Chapter 4

Shipment Consolidation Problem under System-wide Discount

4.1 Problem Description

Suppose an airline offers cargo capacity on its flights between Canada and Europe. The flights are classified into several classes, such as East Coast-Germany, West Coast-Eastern Europe, and so on. Each such class consists of multiple arcs, one for each flight. Airlines charge freight forwarders a base cost per chargeable weight on each flight, and offer a discount factor that depends on the quantity of total cargo on all flights associated with a class. For instance, the cost of shipping a 500 kg package from Toronto to Frankfurt depends on the total weight shipped on the Montreal-Hamburg, Montreal-Berlin, Toronto-Munich and Montreal-Frankfurt routes.

The decision faced by freight forwarders is thus to allocate freight to different flights to make use of that system-wide discount. In other words, a particular freight forwarder needs to determine the load tendered to each flight, in order to minimize the total cost in the presence of the system-wide discount. This problem can also be interpreted as a decision model for a freight forwarder facing multiple system-wide-discount offers from various cargo airlines. Each flight class can be viewed as the flights of a single cargo airline that offers a discount on the total tonnage dispatched over all its network. The decision for a freight forwarder is how to send shipments via these airlines to minimize total shipping cost (by possibly utilizing the system-wide discounts offered by each airline).

4.2 Problem Formulation

To formulate this problem, we begin with the multi-commodity flow model of Cohn et al. (2008), and extend it to include shipment volume and gross weight, prevent shipment splitting, and take advantage of over-declaration. Experienced shippers and freight forwarders will, when appropriate, over-declare the weight of a consolidated shipment to enjoy the discount on the higher-weight segment. (This is the "bumping clause", mentioned in Section 1.1.1.) The model of Cohn et al. (2008) does incorporate a system-wide discount.

Instead of using the common node-pair notation for flights, let us denote flights on an arc by $f \in F$. (Although we used the subscript j to represent a flight in our first problem, we now employ the subscript f to denote a flight in the current problem because Cohn et al. (2008), whose work we are extending, used j to represent a node.) Although f may range from 1 to 1,000, this will yield a more general formulation, as there may exist multiple flight arcs between a node pair in the planning horizon. Let us also denote shipments by $k \in K$, nodes by $n \in N$, and flight classes by $c \in C$, where each flight f appears in exactly one class c denoted by $c(f)$. The discount scheme is of the all-unit type, where R^c represents the set of discount ranges for flight arcs in class c. For each class c and discount range r, l_r^c, u_r^c and p_r^c indicate the lower limit, the upper limit and the discount factor. For each flight f, b_f denotes the base cost (unit cost per kilogram before any discount), while U_f is the total available capacity. As we also take the shipment volume into account, that unit cost b_f for each flight is based on the unit chargeable weight, i.e. the maximum of the gross weight (g_k) and volume weight (v_k). For each shipment k, N_k^- and N_k^+ denote origin and destination node respectively, while N_f^- and N_f^+ are the respective origin and destination nodes of flight f.

The decision variables are:

- x_{rf}= total chargeable weight on flight f, priced according to discount range $r \in R^{c(f)}$.
- $y_{kf} = \begin{cases} 1, & \text{if shipment } k \text{ is loaded on flight } f; \\ 0, & \text{otherwise;} \end{cases}$
- $z_r^c = \begin{cases} 1, & \text{if the total weight on flights in class } c \text{ is in range } r; \\ 0, & \text{otherwise;} \end{cases}$

The mathematical model is formulated as follows:

[SYSWIDE] :

$$\min W = \sum_{f\in F} \sum_{r\in R^{c(f)}} (1-p_r^{c(f)}) b_f x_{rf} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{f\in\{N_f^+=n\}} \sum_{r\in R^{c(n,i)}} y_{kf} - \sum_{f\in\{N_f^-=n\}} \sum_{r\in R^{c(i,n)}} y_{kf} = \begin{cases} -1 \text{ if } n = N_k^- \\ 1 \text{ if } n = N_k^+ \\ 0 \text{ for all other nodes} \end{cases}$$

$$\forall k \in K, n \in N \tag{4.2}$$

$$\sum_{r\in R^c} z_r^c = 1 \quad \forall c \in C \tag{4.3}$$

$$\sum_k v_k y_{kf} \leq \sum_{r\in R} x_{rf} \quad \forall f \in F \tag{4.4}$$

$$\sum_k g_k y_{kf} \leq \sum_{r\in R} x_{rf} \quad \forall f \in F \tag{4.5}$$

$$l_r^c z_r^c \leq \sum_{f\in c(f)} x_{rf} \leq u_r^c z_r^c \quad \forall c \in C, r \in R^c \tag{4.6}$$

$$\sum_{r\in R^{c(f)}} x_{rf} \leq U_f \quad \forall f \in F \tag{4.7}$$

$$0 \leq x_{rf} \leq u_r^{c(f)} \quad \forall f \in F, r \in R^{c(f)} \tag{4.8}$$

$$y_{kf}, z_r^c \in \{0,1\} \quad \forall c \in C, r \in R^c \tag{4.9}$$

The objective function minimizes the total freight cost, priced at the appropriate class discount factor.

Equations (4.2) enforce the balance of flow in the multi-commodity problem. The restrictions (4.3) guarantee that the total chargeable weight for a given class of flights falls in exactly one system-wide discount range. Constraints (4.4) and (4.5) require that the chargeable weight is always the maximum of the volume weight and gross weight. The relations (4.6) indicate that the total chargeable weight declared across all flights in a class and range is either zero or satisfies the upper and lower bounds of the system-wide discount range. Inequalities (4.7) limit the total declared weight on each flight to be within capacity.

Compared to the model of Cohn et al. (2008), we enforce the "unsplittable" requirement on each shipment by adding the binary decision variable y_{kf}, and aggregate the flow variables over all commodities to reduce the number of decision variables. Moreover, the value of the continuous flow variable x_{rf} does not always equal the total chargeable weight of all shipments on the flight: Rather, we may over-declare the chargeable weight of those shipments to enjoy an even greater discount.

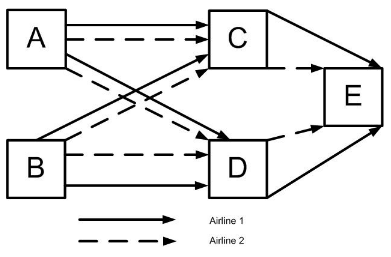
After these modifications, the problem formulation resembles the shipment consolidation problem with a discount range on each flight (we call it the "flight-leg discount version" from now on), presented in Li et al. (2012). The discount range for each flight-class is equivalent to the weight range on each flight. The piece-wise linear cost in the flight-leg discount version can be interpreted as a discount factor in the system-wide discount, as the cost associated with the next-higher weight range is always a certain percent lower than that of the first weight range. To reach that next-higher range, it may be appropriate to over-declare the shipment weight, by analogy to the situation in the flight-leg discount version. Moreover, if we set the number of flight classes equal to the number of flights in the network, with the weight ranges R being the same, the two discount cases become identical.

4.3 An Illustrative Example

To clarify the problem setting, let us consider a small example similar to that of Cohn et al. (2008). A certain airfreight forwarder can tender freight to two carriers (Airlines 1 and 2). Each carrier offers a system-wide discount according to the scheme displayed in Table 4.1, and the six flights displayed in Figure 4.1.

Declared Weight Range	Airline 1 (%)	Airline 2 (%)
0-150	0	0
151-550	20	3
551-700	30	10

Table 4.1: Discount factors for illustrative example



Flight	Cost($ Per kg)	
	Airline 1	Airline 2
AC	2	3
AD	3	5
BC	5	5.1
BD	5	5
CE	5.1	4
DE	5	4

Figure 4.1: Illustrative Example

Now suppose that Shipments 1 and 2, of respective weights 150 and 200 kg, are to be sent from A to E, and Shipments 3 and 4 (each of 125 kg) are to be sent from B to E. Based on the discounts listed in Table 4.1, Shipments 1 and 2 should follow routes AC-CE on Airline 1, and Shipments 3 and 4 follow route BD-DE on Airline 1, for a total cost of

$$[(150 + 200) * (2 + 5.1) + (125 + 125) * (5 + 5)] * (1 - 0.30) = 3489.5$$

Note that this example has 12 flights, 4 shipments, 2 flight classes, and 3 weight ranges in each class of flights. The mathematical formulation requires 36 continuous variables x_{rf}, and 30 binary variables for y_{kf} and z_r^c. For a typical freight forwarder, the planning may involve 100 flights, 400 shipments, 5 flight classes and 3 weight ranges. This corresponds to 300 continuous variables and about 40,000 binary variables.

4.4 Solution Methodology for System-wide Discount

The original work of Cohn et al. (2008) did not provide a dedicated solution methodology to the mathematical model. Here, we present a Lagrangian Relaxation approach to this problem and propose a Lagrangian heuristic to find feasible solutions.

Lagrangian Relaxation

We apply Lagrangian relaxation to Problem [SYSWIDE] by relaxing constraints (4.4) and (4.5) with non-negative multipliers α_f and β_f. This leads to the following subproblem [SYSWIDE-RELAX]:

$$\min \sum_{f\in F}\sum_{r\in R^{c(f)}}(1-p_r^{c(f)})b_f x_{rf} + \sum_{f\in F}\alpha_f(\sum_k v_k y_{kf} - x_{rf}) + \sum_{f\in F}\beta_f(\sum_k g_k y_{kf} - x_f^r)$$

s.t. constraints (4.2), (4.3), (4.6)-(4.9)

[SYSWIDE-RELAX] can be decomposed into two sets of subproblems. The first set is

$$[SP_f^1]: \; obj_f^1 = \min \sum_{r \in R^{c(n,j)}} (1 - p_r^{c(f)}) b_f x_f^r - \sum_f \sum_r \alpha_f^r x_{rf} - \sum_f \sum_r \beta_f^r x_{rf} \quad \forall f$$

$$\text{s.t. constraints (4.3),(4.6)-(4.8)}$$

The second set is

$$[SP_k^2]: \; obj_k^2 = \min \sum_f (\alpha_f v_k + \beta_f g_k) y_{kf} \quad \forall k$$

$$\text{s.t. constraints (4.2) and (4.9).}$$

This decomposition yields a good problem structure, where $[SP_f^1]$ can be solved as a knapsack problem and $[SP_k^2]$ can be solved as a shortest path problem. After obtaining optimal solutions for each subproblem, the sum of the optimal objectives yields a lower bound (LRLB) to the relaxed problem. The best Lagrangian bound is given by: $\max_{\alpha,\beta} \sum_{f \in F} obj_f^1 + \sum_{k \in K} obj_k^2$, which is equivalent to the Lagrangian master problem:

$$[MP]: \max \sum_{f \in F} \theta_f + \sum_{k \in K} \theta_k$$

$$\text{s.t. } \theta_f \le \sum_{r \in R^{c(f)}} [(1 - p_r^{c(f)}) b_f - \alpha_f - \beta_f] x_{rf}^h \quad \forall f, \; h \tag{4.10}$$

$$\theta_k \le \sum_f [(\alpha_f v_k + \beta_f g_k) y_{kf}^h] \quad \forall k, \; h \tag{4.11}$$

$$\alpha_f, \beta_f \ge 0 \quad \forall f \tag{4.12}$$

where $({x_{rf}}^h, {y_{kf}}^h, z_r^{ch})$ are feasible solutions to $[SP_f^1]$ and $[SP_k^2]$. The master problem, subproblems, and multipliers are updated according to algorithms in Figure 4.2.

```
[MP-SP]
Initiate $LRUB = \infty$, $LRLB = -\infty$
    1. Start with a set of $\alpha_f \geq 0$, $\beta_f \geq 0$ (Usually $\alpha, \beta$ are set to be larger than the
       maximum of $b_f$'s)
While $LRUB \neq LRLB$
    2. Solve subproblem $[SP_f^1]$ and $[SP_k^2]$ for each $f$ and $k$ respectively. We get a solution
       of $(x_{rf}^h, y_{kf}^h, z_r^{ch})$ and a lower bound $LRLB^h = \sum_f obj_f^1 + \sum_k obj_k^2$
    3. Update the lower bound $LRLB$=max($LRLB$, $LRLB^h$)
    4. Use $({x_{rf}}^h, {y_{kf}}^h, z_r^{ch})$ to add $|F|$ cuts (4.10) and $|K|$ cuts (4.11) to MP
    5. Solve MP to get a new set of $\alpha, \beta$ and an upper bound $LRUB$
End while
```

Figure 4.2: Algorithm for updating the Lagrange multipliers

The subproblem solution is rarely found to be feasible to the original problem. Section 4.4.1 proposes a heuristic procedure. Section 4.4.2 introduces a branch-and-price procedure to get an exact solution. To obtain a relatively good solution in less computational time, Section 4.4.3 offers a subgradient procedure to update the Lagrange multipliers α and β.

4.4.1 Lagrangian Heuristic

Figure 4.3 illustrates the details of our Lagrangian Heuristic. Solution of $[SP_k^2]$ gives a route for each shipment. This solution is feasible if the total load declared on flight f does not exceed the flight capacity U_f. If that capacity condition is violated for one or more flight arcs, we construct a feasible solution by assigning shipments from over-capacitated flights to less-capacitated ones. Step 1 identifies those flights that have capacity violations.

In Step 2, if there are several alternative flights f, we need to sort the alternatives by their average flight cost. That cost is computed by taking the average of the unit charges over all the system-wide class ranges. We use neither the charge in the lowest weight range nor the one in highest weight range, because it is difficult to predict the range r into which consolidated shipments would fall.

Step 3 is computationally intensive. One could go directly to "adjust the multipliers", instead of performing a "fixing" operation as in Step 3. We have found, however, that Step 3 leads to better solutions in networks having more than a single flight that connects each pair of nodes.

4.4.2 Branch-and-Price

The dual of the Lagrangian master problem [MP] is the Dantzig-Wolfe master problem:

$$\begin{aligned}
\text{[DWM]} : \min \sum_{h=1}^{H^f} \sum_{f \in F} \gamma_f^h \sum_{r \in R^{c(f)}} (1 - p_r^{c(f)}) b_f x_{rf}^h \\
s.t. \; \sum_{h=1}^{H^k} \gamma_k^h v_k y_{kf}^h - \sum_{h=1}^{H^f} \gamma_f^h \sum_{r \in R} x_{rf}^h \leq 0 \quad \forall f, k
\end{aligned}$$

```
Step 1 Let $\{\bar{x}_{rf}, \bar{y}_{kf}, \bar{z}_r^c\}$ be the solutions to $[SP_f^1]$ and $[SP_k^2]$ .

  • Compute $E_f = \max\{\sum_k g_k \bar{y}_{kf}, \sum_k v_k \bar{y}_{kf}\}$ for each flight $f$.

  • If there exists $f$ such that $E_f \geq U_f$, then constraint (4.7) is violated and no feasible
    solution is generated in the current iteration; go to Step 2. Otherwise, go to Step 3.

Step 2 For such flight $\tilde{f}$ that has an infeasible solution in Step 1:
    If there is another flight $f'$ in the same flight class that connects the same origin
        and destination node as $\tilde{f}$,
    then (based on $\{\bar{x}_{rf}, \bar{y}_{kf}, \bar{z}_r{}^c\}$ from Step 1)
        Sort all these shipments that takes flight $\tilde{f}$ by their chargeable weight.
        Load those shipments onto flight $\tilde{f}$ until the flight capacity is reached.
        Load the remaining ones onto $f'$.
    Else Try another flight from a different class or a flight that is part of the path
        between the origin and destination of $\tilde{f}$
Step 3 For each flight class $c \in C$:

  • Let $r'$ be the smallest weight range whose upper limit is larger than or equal to
    $\sum_{f \in c(f)} x_{rf}$.

If $\sum_{f \in c(f)} x_{rf} = 0$,
  then the lowest weight range in this flight class is selected.
Else, For each flight class $c$, weight range $r$,
  If $r = r'$
    If $\sum_{f \in c(f)} (1 - p_r^{c(f)}) b_f x_{rf} > \sum_{f \in c(f)} (1 - p_{(r'+1)}^{c(f)}) b_f l_{r'+1}^c$ and $r' < |R^c|$
      set $x_f^{r'} = 0$ and $z_c^{r'} = 0$
      set $x_f^{r'+1} = l_{k'+1}^c$ and $z_c^{r'+1} = 1$
      $r = r + 1$
    Else
      set $z_r^c = 1$ and $x_{rf} = \bar{x}_{rf}$
  Else
    set $x_{rf} = 0$ and $z_r^c = 0$
```

Figure 4.3: Details of the Lagrangian Heuristic

$$\sum_{h=1}^{H^k} \gamma_f^h g_k y_{kf}^h - \sum_{h=1}^{H^f} \gamma_f^h \sum_{r \in R} y_{kf}^h \leq 0 \quad \forall f, k$$

$$\sum_{h=1}^{H^f} \gamma_f^h = 1 \quad \forall f$$

$$\sum_{h=1}^{H^k} \gamma_k^h = 1 \quad \forall k$$

$$\gamma_f^h \geq 0, \quad h = 1, ..., H^f, \; \forall f$$

$$\gamma_k^h \geq 0, \quad h = 1, ..., H^k, \; \forall f$$

The Dantzig-Wolfe master problem presents an equivalent formulation to [SYSWIDE] when $\gamma_f^h, \gamma_k^h \in \{0, 1\}$. To force this, [DWM] has to be embedded within a branch-and-bound method. As [DWM] is solved through column generation, the result is a branch-and-price approach.

Branching Rules

At each node, we choose a binary variable y_{kf} or z_r^c on which to branch, and branching constraints are appended to the subproblems. There are a few rules to choose the variable to branch on:

1. Branch on y_{kf} before branching on z_r^c. This is attractive because when variable y_{kf} is fixed, the choices of z_r^c are limited.
2. For each y_{kf}, branching is done first on index k before index f. This helps to fathom quite a few nodes. For a given network, there are a great many infeasible subtrees that are easy to identify by looking at y_{kf}. This happens when a certain shipment

k would board a flight f that is scheduled to reach an airport that does not have a connecting flight to k's final destination.

3. When branching on z^c_r, branching is done first on the r that corresponds to the highest weight range. This rule is motivated by the intension to exploit system-wide discounts.

4. Use a depth-first exploration strategy. As running a construction heuristic at each tree node is rather time consuming, it is desired to explore to the lowest level and get a feasible solution quickly.

After branching, two nodes are created based on the branching constraints, and all existing candidate columns are passed to the child nodes.

The Overall Branch-and-price algorithm

The overall branch-and-price algorithm is illustrated in Figure 4.4. It uses a depth-first exploration strategy. At each node of the branch-and-price algorithm, an attempt is made to find feasible solutions to update the incumbent. Since the construction heuristic may be time consuming, we call the construction heuristic only at higher level tree nodes (when $level < \pi$). This helps to reduce the upper bound at earlier stages, therefore increasing the likelihood to fathom nodes sooner.

The whole branch-and-price procedure is terminated when the lower and upper bounds coincide, or all nodes are explored. The procedure is stopped when the two bounds are within 0.05%.

```
[Branch-and-Price]
At root node, initiate $UB = \infty$, $LB = -\infty$

  1. Generate an initial feasible solution (using CPLEX's first feasible) and set the initial
     upper bound $UB$ accordingly.
  2. Start with a set of $\alpha \geq 0$, $\beta \geq 0$ (Usually $\alpha, \beta$ are set to be larger than the maximum
     of $b_f$'s). Use the initial feasible solution as initial columns to construct the master
     problem.

Loop

    Solve the Dantzig-Wolfe problem at the corresponding node as in Figure 4.2, and
    update the current lower bound.
    If problem is infeasible
        fathom current node;
    If current lower bound is greater than $UB$
        fathom current node;
    If we find an integer solution
        Update $UB$ with the solution from master problem, record the integer solution
        if $UB$ is lowered; Terminate the branch-and-price algorithm, when $\frac{UB-LB}{UB} \leq$
        $0.05\%$.
    Otherwise
      A1 Choose the next variable $l$ to branch.
      A2 If $level < \pi$, invoke the Lagrangian heuristic (Section 4.4.1). Update $UB$ if
         the heuristic gives a better upper bound.
      A3 Create two child nodes based on the branching variable and pass all the existing
         candidate columns to the two children according to the branching cut.
      A4 Explore the left child node ($l = 1$).
      A5 Explore the right child node ($l = 0$).

End Loop
```

Figure 4.4: Algorithm for Branch-and-Price

4.4.3 Subgradient Procedure

The branch-and-price algorithm runs very slowly when the problem size is large. In the hope of getting a relatively good solution in a short computational time, we implemented a solution procedure using subgradient optimization. The general idea is to find a good lower bound through manipulating the Lagrange multipliers α and β. Then we use the heuristic in Section 4.4.1 to generate a feasible solution from the best bound we found.

Let $W_{lag}^{\alpha,\beta}$ be the sum of objective values of relaxed problems $[SP_f^1]$ and $[SP_k^2]$. We also define f^* as the best feasible solution found so far in all iterations. Similar to Fisher (1981), the multipliers α_f and β_f are adjusted as follows:

$$\alpha_f^+ = \alpha_f + t(\sum_i v_k y_{kf} - \sum_r x_{rf}) \quad \forall f \tag{4.13}$$

$$\beta_f^+ = \beta_f + t(\sum_i g_k y_{kf} - \sum_r x_{rf}) \quad \forall f \tag{4.14}$$

The step-size function is given by

$$t = \lambda \frac{W^* - W_{lag}^{\alpha,\beta}}{\sum_f [(\sum_k g_k y_{kf} - \sum_r x_{rf})^2 + (\sum_k v_k y_{kf} - \sum_r x_{rf})^2]}$$

We set the initial value of $\lambda_0 = 2$. If the solution W_{lag}^{α} does not improve in 40 consecutive iterations, we reduce the λ_r by half. Starting from an initial set of Lagrange multipliers (α_f, β_f), we solve [SYSWIDE2] to get path designs for each shipment $\bar{y}_{kf}$ and solve $[SP_f^1]$ to determine all declared weights on each flight class $\bar{x_{rf}}$, and each class range assignment $\bar{z}_r^c$. We then get a lower bound $W_{lag}^{\alpha,\beta}$ by substituting $\bar{x}_{rf}$,$\bar{y}_{kf}$,$\bar{z}_r^c$ into the objective function, generating a feasible solution x_{rf},y_{kf},z_r^c from $\{\bar{x}_{rf}$,$\bar{y}_{kf}$,$\bar{z}_r^c\}$. We update the upper bound W^* from that feasible solution. If the gap between $W_{lag}^{\alpha,\beta}$ and W^* is greater than 0.01%, we adjust the multipliers (α_f, β_f) with the subgradient algorithm. This procedure is repeated until the gap between $W_{lag}^{\alpha,\beta}$ and W^* drops below a given threshold. The step-size factor t for adjusting the Lagrange multipliers is cut in half if the best lower bound $W_{lag}^{\alpha,\beta}$ does not improve in 200 iterations.

The overall algorithm is terminated if the solution does not improve in 2000 (*no_improve*) iterations, or an optimal solution is reached before that point, or when the best lower bound improves by less than 0.01% in 100 (T) consecutive iterations. The smaller the values set for *no_improve* and T, the shorter the execution time for the subgradient algorithm.

4.4.4 Generation of Test Cases

We generated random instances that are characterized by four parameters:

- $|N|$: number of nodes.
- $|F|$: number of flights.
- $|K|$: the number of shipments in the network.
- e: a factor representing the minimum number of direct flights between two nodes (in adjacent layers).

Each node in the network belongs to one of four "layers": these are the shipment origins, the exporting hubs, the importing hubs, and the shipment destinations.

We generated twenty-four test cases in Table 4.2 to be used in the following sections. In the case-generation phase, we did not specify the number of flight classes. In later sections, we will vary the number of flight classes ($|C|$) for each network.

Case No.	$\|N\|$	$\|F\|$	e	$\|K\|$
1E1K10	16	48	1	10
1E2K10	16	96	2	10
2E1K30	20	50	1	30
2E2K30	20	100	2	30
3E1K50	30	84	1	50
3E2K50	30	168	2	50
4E1K80	40	100	1	80
4E2K80	40	200	2	80
5E1K60	40	100	1	60
5E2K60	40	200	2	60
6E1K70	40	100	1	70
6E2K70	40	200	2	70
7E1K90	40	100	1	90
7E2K90	40	200	2	90
8E1K100	40	100	1	100
8E2K100	40	200	2	100
9E3K70	40	120	3	70
9E5K70	40	175	5	70
10E1K80	40	120	3	80
10E2K80	40	175	5	80
11E1K90	40	120	3	90
11E2K90	40	175	5	90
12E1K100	40	120	3	100
12E2K100	40	175	5	100

Table 4.2: Cases Generated for the Numerical Tests

4.4.5 Numerical Analysis for Lagrangian Relaxation

The numerical tests were conducted using CPLEX 11.0 on a Windows7 workstation with 1.6GHz i7 CPU and 3GB of RAM. Initially, we applied the branch-and-price algorithm to problem instances generated in Table 4.2. Although branch-and-price did provide an exact solution, the execution time exceeded 30 minutes for all cases except 1E1K10,1E2K10 and 2E1K30. Therefore, we decided to use subgradient optimization methods for the larger problem instances.

Table 4.3 displays results under different numbers of flight classes using Lagrangian relaxation via subgradient optimization. The $\%GAP$ is defined as the percentage difference between the objective computed by the Lagrangian heuristic and that obtained by CPLEX. The average difference between our results and those of CPLEX was below 1% for networks with less than 6 classes of flights.

We define $\%Flight$ in Table 4.3 to denote the percentage of flights being selected in the final solution, relative to the total number of flights in the respective case. One area that Cohn et al. (2008) did not address is the benefit of reducing the number of flights engaged by forwarders. When only two classes of flights are present in the system, just 60% of the available flights will be chosen by freight forwarders. With the availability of such a discount, forwarders not only benefit from the system-wide discount itself, but also tend to load cargo on fewer flights, hence reducing the potential fees for loading or staging. This translates to lower cost in document administration, and in IT integration.

Test Case	$\lvert C\rvert$=2		$\lvert C\rvert$=3		$\lvert C\rvert$=4		$\lvert C\rvert = 5$	
	%GAP	%Flight	%GAP	%Flight	%GAP	%Flight	%GAP	%Flight
1E1K10	0	81.25	0	81.25	0	87.5	0	91.67
1E2K10	0	59.38	0.1	62.5	0.3	73.96	0.1	94.79
2E1K30	0	76	0	80	0	94	0	96
2E2K30	1.2	56	0.8	58	0.1	69	0	90
3E1K50	0	70.24	0	78.57	0.5	78.57	0.1	90.48
3E2K50	3.4	58.93	0.2	58.33	0.3	69.64	0	83.93
4E1K80	0.1	81	0	79	3.6	88	2.2	92
4E2K80	0	57	0.2	60.5	0.2	67	1.6	84.5
5E1K60	0	71.5	0	79	0.7	83.5	0	85
5E2K60	0	69	0.6	72.5	0.6	73	0.1	79
6E1K70	0	80	0	80	0.2	84.5	1.3	89
6E1K70	0.3	74.5	1.1	79	0.2	85	1.2	85
7E1K90	0.5	78.5	0.4	81	0.5	85.5	1.9	84
7E2K90	0.4	64	0.6	68.5	0.5	72.5	2.2	79
8E1K100	1.6	80.5	1.3	89.5	0.8	88	6.4	91
8E2K100	1.2	79	2.9	85	3.8	85	2.3	87.5
9E3K70	0.3	82.5	0.4	86.67	1.4	86.67	1.1	92.5
9E5K70	0	73.14	1.9	74.29	1.3	76	2.3	76
10E3K80	0.4	80.83	0.9	82.5	0.4	84.17	1.5	87.5
10E5K80	1.1	65.14	0	68	0.6	70.86	0.6	69.14
11E3K90	0.6	80	0.9	81.67	1.3	85.83	1.7	92.5
11E5K90	1.2	69.14	1.3	70.29	1	74.29	2	74.86
12E3K100	0.8	57.5	0.2	57.5	0.8	65	2.1	67.5
12E5K100	0.7	73.71	0.7	77.14	3.1	78.86	0.9	80.57
Average	0.58	71.62	0.60	74.61	0.93	79.43	1.32	85.14

Table 4.3: Comparison of Lagrangian Relaxation result for different $|C|$

4.5 Double Discount Case

In this section, we explore *both* a flight-leg and a system-wide discount. In other words, suppose there is a quantity discount c_f^s associated with each flight leg f, and also a quantity discount factor p_r^c associated with flight class c. We use the index s to distinguish the former from the system-wide-discount range, index r. With other parameters remaining the same as in Section 4.2, the following additional notation is defined:

s: index for each quantity-discount range on flight.

c_f^s: base cost per chargeable weight on flight f according to weight range s.

b_f^s: end point of chargeable weight range s for flight f. $b_f^0 = 0$, unless stated otherwise.

The decision variables are modified to:

- x_f^{rs}= total chargeable weight on flight f, priced according to system-wide discount range $r \in R^{c(f)}$ and quantity discount range s.
- $y_{kf} = \begin{cases} 1, & \text{if shipment } k \text{ takes flight } f \\ & \text{range } r \in R^{c(f)} \\ 0, & \text{otherwise;} \end{cases}$
- $w_f^s = \begin{cases} 1, & \text{if combined shipments on flight } f \text{ are priced according to the quantity} \\ & \text{discount range } s \in S; \\ 0, & \text{otherwise;} \end{cases}$
- $z_r^c = \begin{cases} 1, & \text{if the total weight on flights in class } c \text{ is in range } r; \\ 0, & \text{otherwise;} \end{cases}$

The [DOUBLE DISCOUNT] problem is then formulated as:

$$\min \sum_{f \in F} \sum_{r \in R^{c(f)}} (1 - p_r^{c(f)}) \sum_{s \in S} c_f^s x_f^{rs} \tag{4.15}$$

$$\text{s.t.} \sum_{f \in \{N_f^+ = n\}} y_{kf} - \sum_{f \in \{N_f^- = n\}} y_{kf} = \begin{cases} -1 \text{ if } n = N_k^- \\ 1 \text{ if } n = N_k^+ \\ 0 \text{ for all other nodes} \end{cases} \quad \forall k \in K, n \in N \tag{4.16}$$

$$\sum_{r \in R^c} z_r^c = 1 \quad \forall c \in C \tag{4.17}$$

$$\sum_s w_f^s = 1 \quad \forall f \in F \tag{4.18}$$

$$\sum_k v_k y_{kf} \leq \sum_{r \in R^{c(f)}} x_f^{rs} \quad \forall f \in F \tag{4.19}$$

$$\sum_k g_k y_{kf} \leq \sum_{r \in R^{c(f)}} x_f^{rs} \quad \forall f \in F \tag{4.20}$$

$$l_r^c z_r^c \leq \sum_{s \in S} \sum_{f \in c(f)} x_f^{rs} \leq u_r^c z_r^c \quad \forall c \in C, r \in R^{c(f)} \tag{4.21}$$

$$b_f^{s-1} w_f^s \leq \sum_{r \in R^{c(f)}} x_f^{rs} \leq b_f^s w_f^s \quad \forall f \in F, s \in S \tag{4.22}$$

$$x_f^{rs} \geq 0 \quad \forall f \in F, s \in S, r \in R^{c(f)}$$

$$y_{kf}, w_f^s, z_r^c \in \{0, 1\} \quad \forall c \in C, r \in R^c$$

When compared to [SYSWIDE] in Section 4.2, only constraints (4.18) and (4.22) are added. The restrictions (4.18) imply that just one weight range can be used on each flight f. Inequalities (4.22) require that the declared weight should be between the corresponding break points of the weight range. Note that the capacity constraints have been handled implicitly in (4.22), as the upper-weight limit of the highest weight break on a flight is always equal to the booking capacity on that flight.

4.5.1 A Mixed Local Branching Solution

In the double-discount problem, there are three sets of binary variables. The y_{kf} variables are more important in the sense that once they are fixed, the other two are easily known. Based on this, local branching fixes some y_{kf} iteratively to find feasible solutions. When y_{kf}'s are fixed, the problem needs to decide only on the corresponding ranges for the two discount schemes. Therefore, we consider just y_{kf} in the neighborhood exploration. For a given feasible solution $\bar{y}$ to [DOUBLE DISCOUNT], we define the k-OPT neighborhood $N(\bar{y}, \delta)$ of $\bar{y}$ as the set of the feasible solutions of (P) satisfying the additional local branching constraint:

$$\Delta(y, \bar{y}) : \sum_{f} \sum_{k \; and \; y_{kf}=1} (1 - y_{kf}) + \sum_{f} \sum_{k \; and \; y_{kf}=0} y_{kf} \leq \delta \tag{4.23}$$

Therefore, for a given incumbent solution $\bar{y}$, the solution space can be partitioned into a left branch and a right branch according to:

$$\Delta(y, \bar{y}) \leq \delta \text{ (left branch) }, \Delta(y, \bar{y}) \geq \delta + 1 \text{ (right branch)} \tag{4.24}$$

Our implementation is similar to what we did for the unsplittable network consolidation problem in Section 3.3.2, as we consider only the first-level binary variable y_{kf} in the initial neighborhood exploration. After adding constraint (4.23), if the system is not solved to proven optimality within a given time limit, we resort to the variations of second-level variables which are constrained by (4.25) and (4.26):

$$\Delta(w,\bar{w}) : \sum_{f} \sum_{s \text{ } and \text{ } w_f^s=1} (1-w_f^s) + \sum_{f} \sum_{s \text{ } and \text{ } w_f^s=0} w_f^s \le \delta_2 \tag{4.25}$$

$$\Delta(z,\bar{z}) : \sum_{c} \sum_{r \text{ } and \text{ } z_r^c=1} (1-z_r^c) + \sum_{c} \sum_{r \text{ } and \text{ } z_r^c=0} z_r^c \le \delta_3 \tag{4.26}$$

By iteratively increasing the values of δ_2 and δ_3 up to δ, we explored the second-level neighborhoods contained in the first-level neighborhood defined by constraint (4.23). This modification, which caters to two levels of binary variables, yielded excellent results for our test cases, compared to ordinary local branching as in Fischetti and Lodi (2003). According to our computational experience, the choice of δ in the range of $[4, 12]$ is effective in most cases. When we set the δ value too high, it didn't produce the first solution fast enough, which defeats the original purpose of local branching. The second-level and third-level neighborhood size are less sensitive and they are set in the range of $[0.05 * |F| * |K|, 0.08 * |F| * |K|]$ in our implementation.

Another modification we made to the algorithm is a local search (see Figure 4.5) based on the current best feasible solution, before branching on second-level variables, as branching on the second-level variables w_f^s and z_r^c was found to be time consuming. This local search is a simplified version of neighborhood search in Hansen et al. (2006).

We implemented a rather simple node-based tree termination scheme that has the following characteristics, as in Puchinger et al. (2010):

- Trees will be aborted when their total number of created nodes exceeds a given limit.

```
[Local Search]
Loop until the node time limit is reached:
Initiate $nbh = 1$.
Iterate the following steps until $nbh = nbh_{max}$
    1. Generate $\bar{y'}$ at random from the $nbh^{th}$ neighborhood of $\bar{y}$ (denoted by $N_{nbh}(\bar{y})$)
    2. Find the best neighbor $\bar{y'}$ of $\bar{y}$ in $N_{nbh}(\bar{y})$
    3. If $Obj(\bar{y'}) < Obj(\bar{y})$, set $\bar{y}$=$\bar{y'}$ and continue with the search $N_1(\bar{y})$. Otherwise, set
       $nbh$=$nbh + 1$.
End Loop
```

Figure 4.5: Local Search within Local Branching

- Trees will be aborted when the number of created nodes since the last improvement of the best feasible solution found inside the local tree exceeds a given limit.
- When a tree is aborted and the maximum number of local trees is not reached, a new local tree is created with the current best global solution. Based on the result of the last local tree, the new tree will be eventually modified:
  - When a better solution is found since the last tree was created, local branching is restarted with this new solution and the initial local branching parameters.
  - When no new solutions are found, the new local tree is tightened (if the corresponding parameters are set): the number of variables to be fixed is increased, and the value of δ gets modified.

Alternatively, a time limit can be used instead of a tree-node limit at each node level. However, we found that time limit was biased towards smaller sized problem. Therefore, we used the total number of nodes as a limit at each node level, and imposed a time limit at the overall exploration level.

We now compare the results of local branching (with a 15 or 30 minute time limit) with those of running CPLEX for 1 hour. Local branching yields very promising performance according to Table 4.4. The test instances follow the convention of Table 4.2, now adding another dimension: the number of weight ranges on each flight ($|S|$). The number of flight classes is set to either 2 or 3, as differentiated by the digit after character 'C' in the test-case name. We consider the comparison between cases when $|S| = 2$ and $|S| = 4$.

The %GAP column in Table 4.4 measures the difference between local branching and CPLEX, where $\%GAP = \frac{f_{LB} - f_{cplex}}{f_{cplex}} \times 100\%$. A negative (positive) number in this column implies local branching yields a better (worse) result than CPLEX. The last column, *CPLEX Gap*, is the standard CPLEX output that measures the difference between the best feasible objective and the objective of the best node remaining, when the CPLEX operations halt after one hour. The default gap for CPLEX is 0.01%.

Table 4.4 reports the gap between the solution using local branching and CPLEX. Local branching is able to produce solutions to every case combination when two levels of branching are considered. Each case in the right-hand portion of Table 4.4 has four quantity discount ranges. Local branching yields results superior to those of CPLEX in ten cases (within 30 minutes), despite the longer time threshold (one hour) for CPLEX, and ties with CPLEX for seven cases. When the number of quantity discount ranges is reduced to $|S| = 2$, local branching ties with CPLEX in seven cases and surpasses CPLEX in another seven cases. Moreover, local branching shows even greater advantages over CPLEX when

Test Case	$\|C\|$	$\|N\|$	$\|S\|=2$ LB (15min) %GAP	$\|S\|=2$ LB (30min) %GAP	$\|S\|=2$ CPLEX %GAP	$\|S\|=4$ LB (15min) %GAP	$\|S\|=4$ LB (30min) %GAP	$\|S\|=4$ CPLEX %GAP
1C2K10	2	10	0	0	0.01	1.93	0	0.01
1C3K10	3	10	5.04	0	0.01	4.83	0.58	0.01
2C2K30	2	30	0	0	0.01	0	0	0.01
2C3K30	3	30	9.2	0	0.01	0	0	0.01
3C2K50	2	50	0	0	0.01	0	0	0.01
3C3K50	3	50	6.23	0	0.01	2.29	2.29	0.01
4C2K80	2	80	1.1	-0.19	1.08	12.06	2.44	1.02
4C3K80	3	80	7.94	-1.48	1.94	2.41	-2.19	2.86
5C2K40	2	40	2.44	0.82	0.01	8.24	0.61	0.38
5C3K40	3	40	1.68	1.68	0.01	9.38	0.91	0.43
6C2K60	2	60	5.2	0	0.69	9.88	1.29	0.02
6C3K60	3	60	8.65	0.63	0.01	6.73	-0.25	1.14
7C2K70	2	70	8.52	0.55	1.48	0.14	-1.83	2.1
7C3K70	3	70	2.23	-2.3	3.01	-0.98	-0.98	1.46
8C2K90	2	90	4.21	-1.96	2.85	5.49	-2.11	2.93
8C3K90	3	90	-2.41	-2.41	4.33	-0.09	-2.58	3.52
9C2K100	2	100	1.72	-0.12	0.39	0.48	0.29	0.11
9C3K100	3	100	3.02	0.51	0.44	1.89	-0.38	0.56
10C2K110	2	110	4.18	0.99	0.69	1.21	1.07	0.77
10C3K110	3	110	2.25	-1.03	1.38	0.44	-0.56	0.98
11C2K115	2	115	1.8	1.15	0.47	1.29	-0.2	1.16
11C3K115	3	115	4.11	0.84	0.29	-0.03	-0.13	0.78
12C2K120	2	120	3.17	1.33	0.98	6.81	3.02	2.94
12C3K120	3	120	2.03	2.03	1.22	6.94	1.88	0.94

Table 4.4: Comparison of Local Branching with CPLEX

the number of shipments in the test cases increases in Table 4.4. We also want to highlight that local branching gives better results when there are *three* flight classes, compared to the two-class cases.

The choice of first-level neighborhood size δ also plays a vital role in solution quality. The local branching constraint possibly defines a very large neighborhood. Given our problem with $n = |K| \times |F|$ first-level variables, the local tree includes all neighbors with a Hamming distance not larger than δ, so the actual search tree contains $\binom{n}{\delta} = \frac{n!}{(n-\delta)!\delta!}$ neighbors with a Hamming distance of δ. Table 4.5 reports the computational performance based on different first level neighborhood size δ. The $\%GAP$ is measured in the same way as in Table 4.4. To make our analysis less biased, with each combination of $|C|$, $|K|$ and $|F|$, we generated four instances. The $\%GAP$ then represents the average of those four. We chose a 15-minute threshold because, for smaller instance size, the algorithm will not make much more of an improvement after 30 minutes. This is the best time interval for comparison.

$\|C\|$	$\|K\|$	$\|F\|$	%GAP for neighborhood size for 15 min			
			4	8	12	16
2	60	84	3.42	1.69	4.77	6.09
2	60	168	4.08	1.34	0.98	6.20
3	60	84	2.66	0.99	0.92	2.64
3	60	168	4.85	0.78	1.03	6.31
2	80	84	3.10	1.89	0.57	3.91
2	80	168	2.99	0.94	1.02	3.01
3	80	84	5.91	2.19	1.23	4.30
3	80	168	4.06	1.80	2.02	1.95
2	120	84	5.96	3.08	4.73	3.40
2	120	168	6.74	3.91	3.86	4.21
3	120	84	5.35	4.30	4.06	5.02
3	120	168	5.78	2.77	1.70	2.05

Table 4.5: Comparison of Local Branching on the Choice of δ

The trend in Table 4.5 reveals that when the neighborhood size is small ($\delta = 4$), the algorithm is able to control the gap below 4.5% if the instance size is relatively small. The large-neighborhood-size setting ($\delta = 16$) result in a larger average gap. When the variables involved in the problem increase, a smaller neighborhood size always leads to traps in local minima, and no longer produces a good result. The best results shift to settings where $\delta = 12$ or 16.

4.5.2 Practical Benefits of the Double Discount

To assess the effectiveness of the double-discount scheme, we compare it to the no-discount, flight-leg, and system-wide discounts. The findings are displayed in Table 4.6, where the cost savings and the percentages of the total number of flights used are exhibited. The data in the "No Discount" column are generated by assigning a unit cost equal to the average costs of the flight-leg and the system-wide discounts. There are no bumping clauses nor system-wide discounts on any flight legs. The average total discount over all flights is the same for the flight-leg and the system-wide discounts.

Case	$C\|$	No Discount		Flight-Leg Discount		System-Wide Discount		Both	
		Total Cost($)	% of flights used	Total Cost($)	% of flights used	Total Cost($)	% of flights used	Total Cost($)	% of flights used
1C2E2K10	2	17,620	79.17	16,844	65.63	13,405	75.00	12,683	64.58
1C3E2K10	3	17,620	79.17	16,844	65.63	14,421	78.95	12,991	63.54
2C2E2K30	2	43,181	83.00	41,070	64.00	36,991	67.47	33,581	55.00
2C3E2K30	3	43,181	83.00	41,070	64.00	37,424	69.88	34,760	55.00
3C2E2K50	2	70,014	75.00	51,582	61.90	49,744	78.57	46,430	56.55
3C3E2K50	3	70,014	75.00	51,582	61.90	48,590	77.78	46,886	57.14
4C2E2K80	2	113,680	83.50	89,474	69.50	85,240	68.26	83,611	59.00
4C3E2K80	3	113,680	83.50	89,474	69.50	87,504	72.46	83,554	59.50

Table 4.6: Comparison of Double Discount with No Discount and Single Discounts

From Table 4.6, we observe a significant drop in the average percentage of flights used for the flight-leg discount and the system-wide discount, compared to the no-discount cases. As expected, the double-discount achieves the best results. However, the further reductions of the double-discount from the flight-leg discount or system-wide discount cases are smaller percentage than those reductions from no-discount to single discount. That is because it is hard for declared shipments to reach the last segments on a flight-leg discount weight range and on a class weight range at the same time. This is also a reason for the increased cost when there are more flight classes. As supported by Table 4.6, the overall cost is 0.5% higher when there are three flight classes compared to just two.

4.6 Summary

In this chapter, we have proposed two mathematical models and corresponding solution procedures for the shipment consolidation problem under two discount schemes: a system-wide discount and a double-discount.

For the system-wide discount, we extended an existing model due to Cohn et al. (2008) by including the volume weight, the possibility of weight over-declaration, and the requirement that a given shipment cannot be divided. We then proposed a column generation framework to solve it. Under such a discount, forwarders not only benefit from the system-wide discount itself, but also tend to engage with fewer airlines, thus reducing administrative fees.

The second discount scheme combined both the system-wide and the flight-leg discounts. Using a mathematical programming model, we presented a local branching approach to solve it and discussed some of the model's practical benefits.

Chapter 5

Consolidation Problem with Pivot Weight

5.1 Problem Description

In this chapter, we study a freight consolidation problem variant where shipments are consolidated into ULDs. An airfreight forwarder is faced with the decision to allocate a total of n shipments. Each shipment $i \in I$ (I is the set of shipments), with gross weight g_i is to be allocated to a particular ULD $j \in J$ (J is the set of reserved ULDs), subject to a capacity limitation. Suppose there are $|J|$ ULDs, each with a fixed reservation cost f_j, a pivot capacity U_j, an extra pivot capacity U_j^E, an under-pivot rate c_j and an over-pivot rate c_j^E. The ULD thus has a total capacity of $U_j + U_j^E$. Given customer demand, the airfreight forwarder needs to decide on which ULDs to select. For those ULDs chosen, the forwarder also needs to determine which shipments will be loaded in each, in order to minimize the total cost.

5.2 Problem Formulation

We use binary decision variables x_{ij} and z_j, where x_{ij} takes value 1 if shipment i is assigned to ULD j, and 0 otherwise; z_j takes value 1 if ULD j is used and 0 otherwise; and continuous variables y_j^E to denote the additional capacity beyond the pivot weight for ULD j. The air-cargo consolidation problem with pivot weight is modeled as:

$$\text{[ACPW]: } \min \sum_j f_j z_j + \sum_i \sum_j g_i c_j x_{ij} + \sum_j c_j^E y_j^E \tag{5.1}$$

$$\text{s.t.} \sum_j x_{ij} = 1 \quad \forall i \in I \tag{5.2}$$

$$\sum_i g_i x_{ij} \leq U_j z_j + y_j^E \quad \forall j \in J \tag{5.3}$$

$$y_j^E \leq U_j^E z_j \quad \forall j \in J \tag{5.4}$$

$$x_{ij} \in \{0,1\},\ z_j \in \{0,1\},\ y_j^E \geq 0 \quad \forall i, j$$

The objective (5.1) minimizes the fixed reservation cost plus the under-pivot and over-pivot costs. Constraints (5.2) require that each shipment be assigned to exactly one ULD. Constraints (5.3) and (5.4) model the pivot capacity and over-pivot capacity for each ULD j. Li et al. (2009) showed that this problem is NP-hard, as it can be reduced to the well-known 3-partition problem. In the following section, we propose a branch-and-price algorithm to solve the problem.

5.3 Branch-and-Price

By relaxing constraint (5.2), we obtain the following subproblem:

$$\text{[ACPW-Relax]: } \min \sum_j f_j z_j + \sum_i \sum_j g_i c_j x_{ij} + \sum_j c_j^E y_j^E + \sum_i \lambda_i (1 - \sum_j x_{ij}) \tag{5.5}$$

$$\text{s.t.} \sum_i g_i x_{ij} \le U_j z_j + y_j^E \quad \forall j \tag{5.6}$$

$$y_j^E \le U_j^E z_j \quad \forall j \tag{5.7}$$

$$x_{ij} \in \{0,1\},\ z_j \in \{0,1\},\ y_j^E \ge 0 \quad \forall i,j$$

Note that since we relaxed an equality constraint, the multiplier λ_i is unrestricted in sign. The subproblem can be decomposed to $|J|$ subproblems as follows:

$$[SP_j] : W_j = \min f_j z_j + \sum_i (g_i c_j - \lambda_i) x_{ij} + c_j^E y_j^E$$

$$\text{s.t.} \sum_i g_i x_{ij} \le U_j z_j + y_j^E \tag{5.8}$$

$$y_j^E \le U_j^E z_j \tag{5.9}$$

$$x_{ij} \in \{0,1\} \ \ \forall i,\ z_j \in \{0,1\},\ y_j^E \ge 0$$

$[SP_j]$ is a 0/1 knapsack problem with one continuous variable y_j^E and an additional constraint (5.9), corresponding to each ULD j. The Lagrangian bound is given by $\sum_j W_j + \sum_i \lambda_i$. The best Lagrangian bound is thus $\max_\lambda \sum_j W_j + \sum_i \lambda_i$, which is equivalent to the Lagrangian master problem:

$$\begin{aligned}
\text{[MP]: max } & \sum_i \lambda_i + \sum_j \theta_j \\
\text{s.t. } & \theta_j + \sum_i \lambda_i x_{ij}^h \leq f_j z_j^h + \sum_i g_i c_j x_{ij}^h + c_j^E y_j^{Eh} \quad \forall h,\ \forall j \qquad (5.10)
\end{aligned}$$

Here (x_{ij}^h,y_j^{Eh},z_j^h) is a feasible solution to $[SP_j]$, where $1 \leq h \leq H_j$ denotes the iteration number. The master problem yields a Lagrangian upper bound $LRUB$, while the subproblems produce a Lagrangian lower bound $LRLB = \sum_i \lambda_i + \sum_j SP_j(\lambda)$. The master problem, subproblem, and multipliers are updated iteratively according to the following steps in Figure 5.1.

A solution to the subproblem is rarely found to be feasible for the original problem. Section 5.3.1 proposes a column generation procedure to get an exact solution.

5.3.1 Column Generation And Branch-and-Price

The dual of [MP] is the Dantzig-Wolfe Master problem:

$$\text{[DWM]: min } \sum_{h=1}^{H_j} \sum_j (f_j z_j^h + \sum_i g_i c_j x_{ij}^h + c_j^E y_j^{Eh}) \alpha_j^h$$

```
Initiate $LRUB = \infty$, $LRLB = -\infty$
    1. Start with a set of $\lambda_i$ (For example $\lambda_i$ are set to be greater than the maximum of
       $c_j g_i + c_j^E + f_j$)
While $LRUB \neq LRLB$
    2. Solve subproblem $[SP_j]$ for each $j$. We get a solution ($x_{ij}^h$,$y_j^{Eh}$,$z_j^h$) and a lower bound
       $LRLB^h = \sum_j W_j^h + \sum_i \lambda_i$
    3. Update the lower bound $LRLB$=max($LRLB$, $LRLB^h$)
    4. Use ($x_{ij}^h$,$y_j^{Eh}$,$z_j^h$) to add $|J|$ cuts (5.10) to [MP]
    5. Solve [MP] to get a new set of $\lambda_i$ and an upper bound $LRUB$
End while
```

Figure 5.1: Algorithm for updating the Lagrange multipliers

$$\text{s.t.} \quad \sum_j \sum_{h=1}^{H_j} x_{ij}^h \alpha_j^h = 1 \quad \forall i \tag{5.11}$$

$$\sum_h^{H_j} \alpha_j^h = 1 \quad \forall j \tag{5.12}$$

$$\alpha_j^h \geq 0, \quad h = 1 \dots H_j$$

The Dantzig-Wolfe master problem presents an equivalent formulation to [ACPW] when $\alpha_j^h \in \{0,1\}$. To force this, [DWM] has to be embedded within a branch-and-bound method. As [DWM] is solved through column generation, the result is a branch-and-price approach.

At each node of the branch-and-price algorithm, the Lagrangian dual is solved as in Figure 5.1. Then a feasible solution is obtained as in Section 5.3.4. That feasible solution is used to update the incumbent. Depending on these bounds, a node is either fathomed, or further explored by branching. The procedure is repeated until all nodes are fathomed.

5.3.2 Branching

Theoretically, we can branch on either x_{ij} or z_j. However, after some initial testing, we found that branching first on z_j gave additional feasible columns at an early stage of the branch-and-price process. Therefore, we branch first on z_j, followed by x_{ij}. When branching on x_{ij}, we try to branch on the j index before the i index. The two-level branching strategy is illustrated in Figure 5.2.

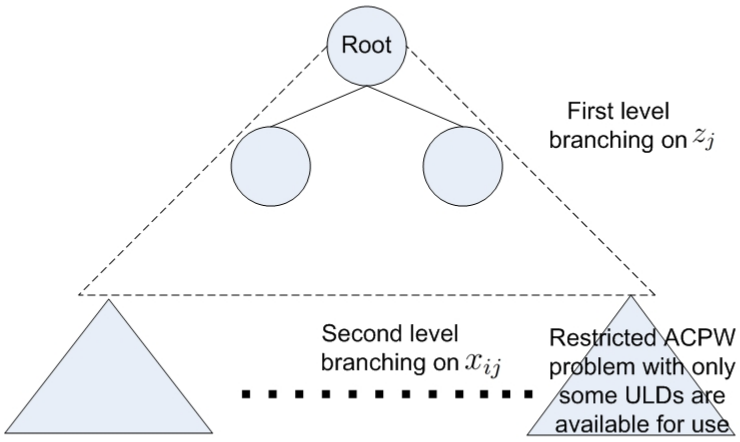

Root
First level branching on z_j
Second level branching on x_{ij}
Restricted ACPW problem with only some ULDs are available for use

Figure 5.2: Two-level Branching Strategy

A depth-first strategy will explore the choice where only one ULD is open, which is unlikely to yield a feasible solution. Recall that we need a good feasible solution to update the incumbent and increase the likelihood of successfully fathoming the subsequent child nodes. Therefore, branching first on j will lead to a solution with all ULDs being utilized.

This will result in a feasible solution, which can be used as a good upper bound for fathoming the right side of the tree.

The preceding strategy resembles the branching strategy in facility location problems. There, it is always better to branch first on whether facility is "open" or "closed". Once we have decided whether to use a given ULD or not, we can then decide on the assignment of shipments to ULDs. Moreover, for each z_j, we branch on those j with smaller values of $\frac{f_j + c_j * U_j}{U_j}$, as the ULDs with lower unit cost are more likely to be used. For the binary variables x_{ij}, we give branching priority to the ones with the greatest fractional values.

Branching is halted when one of the following three conditions is met:

- The Lagrangian lower bound exceeds the incumbent
- "Closing" any ULD will result in an infeasible solution
- All nodes created by the branching rule have been searched

5.3.3 Generating an Initial Feasible Solution

A basic feasible solution can be obtained from the relaxation of the original [ACPW] formulation. We approximate the primal integer solution to [ACPW] by rounding its LP, denoted by:

$$\text{[LP-ACPW]: } \min \sum_j f_j z_j + \sum_i \sum_j g_i c_j x_{ij} + \sum_j c_j^E y_j^E$$

$$\begin{aligned}
\text{s.t.} \sum_j x_{ij} &= 1 \quad \forall i \in I \\
\sum_i g_i x_{ij} &\le U_j z_j + y_j^E \quad \forall j \in J \\
y_j^E &\le U_j^E z_j \quad \forall j \in J \\
0 \le x_{ij} \le 1,\ 0 \le z_j &\le 1,\ y_j^E \ge 0 \quad \forall i, j
\end{aligned}$$

The next step can be viewed as a two-step approach similar to that of Akeb et al. (2011):

- Once [LP-ACPW] is solved, fractional x_{ij} and z_j are rounded. The fractional variable that has the largest value is rounded first.
- The reduced problem that remains after the preceding step is subject to a similar procedure, until no further rounding is needed.

5.3.4 Lagrangian Heuristic

When we solved the Dantzig-Wolfe problem at each tree node, we often obtained infeasible solutions from the subproblems. We thus need a quick heuristic that can generate a feasible solution based on the solution to the subproblem. The infeasible solutions above usually had some shipments being assigned to multiple ULDs, while other shipments were not assigned to any. Therefore, we use the following Lagrangian Heuristic:

1. For any shipment assigned to more than one ULDs; For each of those j ULDs, evaluate the ratio $\frac{f_j+c_j*U_j}{U_j}$ value, retain the shipment in the ULD with the minimum ratio. Remove that shipment from the remaining ULDs.

2. Treat each ULD as two knapsacks, one with a capacity of U_j and the other with capacity of $U_j + U_j^E$. (These are respectively type (i) and type (ii).)

3. Sort all shipments in descending order of their weight

4. Update each ULD's available capacity: $U_j{}'{=}U_j{-}\sum_{i\ loaded\ into\ j} g_i$;

5. Sort all ULDs in ascending order of $\frac{f_j+c_j*U_j'}{U_j'}$ if they are type (i) knapsacks, or in ascending order of $\frac{f_j+c_j*U_j+c_j^E*U_j^E}{U_j+U_j^E}$ for type (ii) knapsacks. Only one type can be used at a time.

6. Fill the ULDs with shipments in the sorted order.

7. If the total weight in the ULD does not exceed U_j, fill that ULD with shipments until the total weight contained just exceeds the pivot weight U_j. Let r denote the shipment that first causes the ULD's load to exceed the pivot weight. Let S represent the solution with shipment r, and S' represent the solution without shipment r. Choose S or S', whichever has lower cost. (See Figures 5.3(a) and 5.3(b) for an illustration.)

8. If there still remain any shipments not allocated to ULDs, return to Step 4. Otherwise, stop.

The preceding algorithm is similar to the profit-density greedy approach to the knapsack problem, where $\frac{f_j + c_j * U_j}{U_j}$ is equivalent to the profit-density ratio for each shipment. The heuristic for Case 2 is derived based on the following observation: When the binary requirement is relaxed for each binary variable in the knapsack problem, the optimal solution is still binary except for the last item in the knapsack.

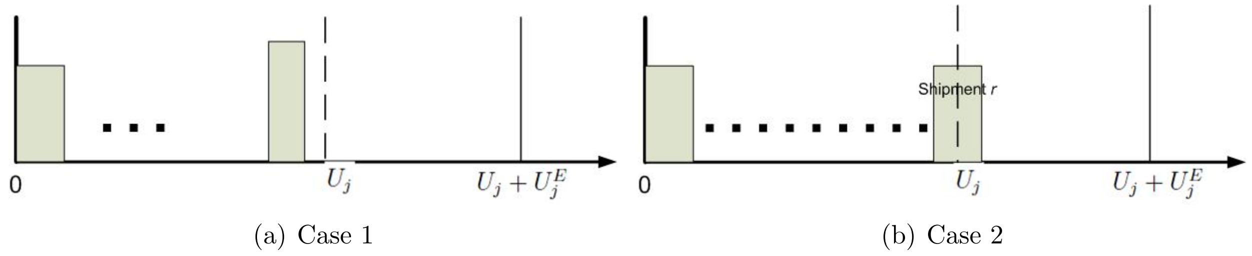


Figure 5.3: The Two Cases of Step 7 in Lagrangian Heuristic

5.4 A Best-Fit Decreasing Heuristic

The branch-and-price procedure in Section 5.3 is usually very slow to converge. The airfreight industry requires rapid decisions in operational planning. Therefore, we require a fast and accurate heuristic for the ACPW. The heuristic we propose in this section is motivated by heuristics for the bin-packing and the knapsack problems.

We adopt the best-fit decreasing heuristic (BFDH) based on profit density. Kan et al. (1993) suggest a class of generalized greedy algorithms for the $\{0, 1\}$ multi-knapsack problem. Items are selected according to decreasing ratios of their profit and a weighted sum of their requirement coefficients.

The BFDH first sorts all shipments according to the non-increasing order of their gross weight. For each shipment, the algorithm initially attempts to load it in the "best" already-selected ULD, which is that ULD with the minimum "adjusted-unit-cost" ratio, defined as $\frac{f_j+c_j*U_j}{U_j}$ if no over-pivot capacity is used; or $\frac{f_j+c_j*U_j+c_j^E*U_j^E}{U_j+U_j^E}$ otherwise. If the shipment cannot be loaded on an already-selected ULD, a new ULD is chosen. Once a shipment is assigned to a ULD, the "best" ULD is re-computed according to the space remaining in each ULD. Due to the existence of over-pivot capacity, the calculation of adjusted-unit-cost ratio has to take c_j^E and U_j^E into account. The detailed algorithm is shown below.

```
1. Initialize $S = \{\emptyset\}$. Sort all shipments $i$ in decreasing order of their gross weight.

2. Set $U_j'=U_j$, $R_j=\frac{f_j+c_j*U_j'}{U_j'}$ for each ULD.

3. For all $i \in I$ (assumed to be in descending order of gross weight)

    3.1 for all $j \in J$

            if $g_i > U_j' + U_j^E$ go to the next ULD
            else if $g_i > U_j'$ set $R_j = \frac{f_j+c_j*U_j'+c_j^E*U_j^E}{U_j'+U_j^E}$
            end if

    3.2 sort all ULDs in ascending order of $R_j$. Let $j'$ be the first ULD in the ordered
        list, and fill $i$ into $j'$. Set $U_j'=U_j - g_i$,$S = S \cup \{j'\}$.

4. For all $j \in S$ do

4.1 set $M_j = \sum_{i\ i\ loaded\ in\ j} g_i$
```

```
4.2 for all $k \in J \backslash S$ do
        if $U_k > M_j$ and $c_j < c_k$ then
        move all the items from $j$ to $k$
        $S = S \backslash j \cup \{k\}$
        end if
5. Call LocalSearch($j$, $size(j)/3$), described next.
```

The Local Search routine is as follows:

Loop until the local time limit is reached:

Initiate $nbh = 1$.

Iterate the following steps until $nbh = nb_{max}$

1. For the given ULD j, generate $\bar{x}'_{ij}$ at random from the nbh^{th} neighborhood of $\bar{x}_{ij}$ (denoted by $N_{nbh}(\bar{x}'_{ij})$), where $j \neq j'$.
2. Find the best neighbor $\bar{x}'_{ij}$ of $\bar{x}_{ij}$ in $N_{nbh}(\bar{x}_{ij})$
3. If $Obj(\bar{x}'_{ij}) < Obj(\bar{x}_{ij})$, set $\bar{x}_{ij}$=$\bar{x}'_{ij}$ and continue the search within $N_{nbh}(\bar{x}_{ij})$, where $nbh = 1$. Otherwise, set nbh=$nbh + 1$.

end

Our calculation of the adjusted-unit-cost ratio is inspired by the item-selection rule for knapsack problems. We thus select bins according to the non-increasing order of their unit cost/volume ratios, and in the non-decreasing order of their volumes when the unit costs are equal.

5.5 A Local Branching Heuristic

In this section, we first apply the multi-level-variable branching extension proposed in Chapters 3 and 4 to the ACPW problem. Based upon handling those multi-level variables, we introduce new extensions to improve the existing local branching framework.

5.5.1 The Basic Local Branching Algorithm

Given a feasible solution $\bar{z}$ of ACPW and a positive integer parameter k, the k-OPT neighborhood $N(\bar{z}, k)$ of $\bar{z}$ is the set of feasible solutions of ACPW satisfying the additional local branching constraint (which is also called the Hamming Distance constraint):

$$\Delta(z, \bar{z}) : \sum_{j \; and \; \bar{z}_j=1} (1 - z_j) + \sum_{j \; and \; \bar{z}_j=0} z_j \leq k \tag{5.13}$$

Therefore, for a given incumbent solution $\bar{z}$, the solution space can be partitioned into a left branch and a right branch according to (5.14):

$$\Delta(z, \bar{z}) \leq k_1 \text{ (left local tree) }, \Delta(z, \bar{z}) \geq k_1 + 1 \text{ (right local tree)} \tag{5.14}$$

According to computational experience, the choice of k in the range of $[6, 12]$ is effective in most cases. In addition to the framework proposed by Fischetti and Lodi (2003), we consider only the first-level binary variable z_j in the initial neighborhood exploration. After adding constraint (5.13), if the system is not solved to proven optimality within a given time limit, we resort to the variation of second-level variables which is constrained by (5.15):

$$\Delta(x, \bar{x}) : \sum_{i} \sum_{j \; and \; \bar{x}_{ij}=1} (1 - x_{ij}) + \sum_{i} \sum_{j \; and \; \bar{x}_{ij}=0} x_{ij} \leq k_2 \tag{5.15}$$

By iteratively increasing the value of k_2 up to k_2^{max}, we explore the second-level neighborhoods which are contained in the first-level neighborhood defined by constraint (5.13). This modification caters to two levels of binary variables, compared to the original local branching in Fischetti and Lodi (2003).

Although the multi-level-variable strategy gained success in the problems of Chapters 3 and 4, it does not deliver good performance for the ACPW problem. In the previous two research problems, when the first level variables are fixed, the feasible solution space is reduced significantly (as feasible routes are highly dependent on the choice of flights). In ACPW, that feasible solution space is relatively large compared to the number of first-level variables (z_j), and a promising left-subtree does not always lead to an overall optimal solution.

To enhance our local branching implementation, two extensions have been added to the standard local branching algorithm. The first eliminates the restriction of sequential execution by allowing us to create new local trees before the previous one(s) are finished. This permits us to start from multiple initial solutions created by the steps in Sections

5.3.3 and 5.3.4. The second extension tries to reduce the subproblem complexity by fixing those variables that are less likely than others to change in the optimal solution.

5.5.2 Local Branching with Multiple Trees

We could easily have multiple initial solutions by the methods in Sections 5.3.3 and 5.3.4. These solutions will generate a number of local trees from the root node. We leverage on the multi-thread feature of Java to implement the pseudo-concurrent exploration of several local trees. This introduces some diversifications and avoids the final solution being caught in a local optimum. Similar to the implementation of Lichtenberger (2005), at each node of the local branching tree, there is a single pool for subproblems where all local tree nodes are stored. Each local tree generated by the incumbent tree node is assigned a new parallel thread. We give promising nodes more execution time than others by assigning a tree exploration time-threshold proportional to their lower bound, i.e. local trees with a better lower bound will get more execution time.

This “mimics” the parallel exploration by a modification to the standard local branching algorithm: Before a local tree is completely solved, the right (inverse) local-branching constraint for the rest of the search tree remains inactive. When this local tree is prematurely terminated, that right constraint is removed from all future local trees. The detailed implementation is illustrated in Figure 5.4, where three initial solutions are considered. We use “pseudo-concurrent” here because CPLEX allows the building of different local tree nodes with multiple duplicates of environment (CPXENV and CPXLP) in the implementation. However, under the academic license, CPLEX restricts the implementation from concurrently operating on the same object (as we would theoretically do in parallel programming).

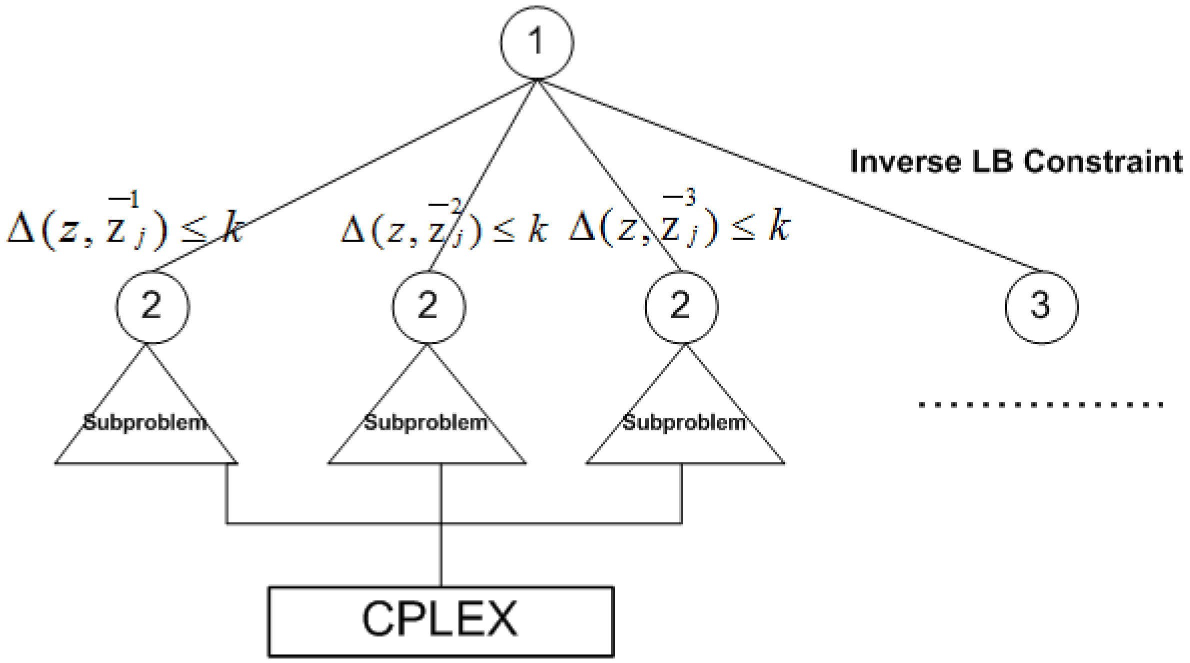

1
Inverse LB Constraint
$\Delta(z,\bar{z}^1_j) \le k$
$\Delta(z,\bar{z}^2_j) \le k$
$\Delta(z,\bar{z}^3_j) \le k$
2
2
2
3
Subproblem
Subproblem
Subproblem
CPLEX

Figure 5.4: Local Branching with Multiple Local Trees

As indicated by Lichtenberger (2005), this kind of pseudo-concurrent exploration will likely lead to a duplication of effort in some cases. Since it is not known in advance which trees will actually be completely solved, right-tree constraints cannot be considered until all the left-tree constraints have been dealt with. When a local tree is prematurely terminated, no information about this local tree (except feasible solutions found so far) can be further utilized: the neighborhood defined by this tree cannot be excluded from future local trees because it still may contain the optimal solution. Therefore, whenever we branch on z_j, the right-tree constraint of the less-promising node is generated together with the right tree of the promising one.

5.5.3 A Modified Relaxation-Induced Neighborhood Search Heuristic

In the Relaxation-Induced Neighborhood Search (RINS) proposed by Danna et al. (2005), it is assumed that variables having the same integer value in the incumbent solution and in the LP relaxation are likely to be set to their optimal value. We apply this idea and embed it in our local branching implementation. Thus, at each node of the local branching:

1. Fix a subset of variables that have the same values in the incumbent and in the continuous relaxation.

2. Focus on the fractional variables. An MIP subproblem is solved on the remaining variables within a given time limit.

3. If a better solution can be found, it is passed to the global MIP-search after a solution to the MIP subproblem is found.

4. Otherwise, explore outside the neighborhood of the incumbent in the global MIP.

Therefore, at each tree node, Steps 1, 2 and 3 correspond to the left local tree (in basic local branching), and act as an *intensification* step, where the more promising factional variables are explored further. At the same time, the "less promising" integral variables of the current

LP optimum are ignored. This variable-fixing strategy explores a neighborhood both of the incumbent and of the continuous relaxation, compared to reducing the neighborhood size by decreasing the value of k in basic local branching framework. RINS not only improves incumbents of poor quality (because it is guided by the continuous relaxation), but also likely enhances robustness when faced with a loose relaxation. With the MIP subproblem potentially difficult to solve, we will also impose a time limit for RINS at each node. Therefore, this intensification step will not plunge into any difficult local optima.

Our implementation follows the framework proposed by Lichtenberger (2005), we randomly selects from the set of all variables having the same integer values in the integral and the LP solution. The number of fixed variables is given relative to the total number of variables in the subproblem. In the following, let M_1 denote the indices of variables fixed to one, and M_0 the indices of variables fixed to zero. ($M_0 + M_1$ thus equals the total number of variables to be fixed). The variable-fixing process in Step 1 can be done directly using cut (5.16).

$$\sum_{j \in M_1} (1 - z_j) + \sum_{j \in M_0} z_j = 0 \tag{5.16}$$

For the right local branching tree in Step 4, suppose a solution is feasible outside the left local tree. It is then true that either:

- At least one binary variable in the set of "less promising" variables has flipped its value, or
- the Hamming distance of the new solution to the incumbent solution is greater than k.

The first condition, which is complementary to (5.16), can be represented by cut (5.17). The second condition can be combined with (5.17) into a new cut (5.18):

$$\sum_{j \in M_1} (1 - z_j) + \sum_{j \in M_0} z_j > 0 \tag{5.17}$$

$$\Delta(z, \bar{z}) > k - k \left[\sum_{j \in M_1} (1 - z_j) + \sum_{j \in M_0} z_j \right] \tag{5.18}$$

where $\Delta(z, \bar{z})$ denotes the Hamming distance between the initial solution $\bar{z}$ and the current solution z_j as defined in Equation (5.13). When one of those fixed variables flips, the right-hand side will be less than or equal to zero. In this case, constraints (5.18) are satisfied even if the Hamming distance is smaller than k (since that Hamming distance is always non-negative).

In the ACPW problem, variables are more likely to take the same values in the incumbent and relaxation, compared to other consolidation problems involving many routing decisions. Moreover, ACPW has a loose linear relaxation compared to those other problems. Therefore, ACPW is a more suitable application for a RINS-orientated local branching.

5.6 Computational Analysis

In this section, the computational performance of the proposed model and solution method is evaluated. All proposed algorithms and the heuristic are coded in Java with CPLEX 11 as the back-end optimization solver on a laptop with i-7 Quad Core processors running at 1.6 GHz and 3GB of memory.

As the ACPW problem is relatively new and the business data from air freight forwarders is confidential, we generated our own test cases to compare the performance of the proposed approaches. Each such case is characterized by the number of shipments, the number of ULDs, a range in which shipment weight is uniformly distributed, as well as a range in which the pivot weight of ULDs is uniformly distributed. The shipment weight is uniformly distributed in the interval [100,300]. The percentage by which U_j^E exceeds the pivot weight U_j ($\%U_j^E$ vs U_j) is also taken into consideration in test case generation, where the default value for $\%U_j^E/U_j$ is 10% ($U_j^E = 10\%U_j$). The default ratio of $\frac{c_j^E}{c_j}$ is set to 1.2 unless otherwise specified.

Note that the sum of the maximum weights of all ULDs is set to be larger than the total weight of all shipments, while the sum of their pivot weights differs from the total weight of all cargo by a random amount in the interval $[0, 0.02\sum_j U_j]$. This follows the same assumption of Li et al. (2009), based on the fact that the cargo weight of a ULD in an ideal plan should be around its pivot weight; the forwarder tries to achieve this when reserving the capacity.

5.6.1 Performance of the Branch-and-Price Algorithm

For the branch-and-price algorithm, we set a time limit of 7200 seconds. Computational time is reported in column CPU if the algorithm is able to find the optimal solution within the time threshold. The gap(%) is defined as the average gap between the best upper and lower bounds found within the given time limit, i.e. $gap = \frac{UB-LB}{LB}\%$.

Table 5.1 shows that the branch-and-price approach is quite satisfactory when the number of bins is small, as the algorithm is able to solve instances of up to 150 shipments. 25 instances out of 32 are solved to optimality within the two-hour limit. However, when the number of ULDs increases to 16, the algorithm could not reach an optimal solution within two hours and the gap between the lower and upper bounds remained relatively large.

n	N	$U_j^E/U_j\%$			
		10%		30%	
		gap(%)	CPU	gap(%)	CPU
4	20	0	2	0	4
	40	0	71	0	194
	100	0	1917	0	2068
	150	0	3044	0	996
8	20	0	205	0	179
	40	0	426	0	802
	100	0	1703	0	780
	150	0	3077	0	6023
12	20	0	443	0	325
	40	0	2438	0	2201
	100	2.2	7200	3.1	7200
	150	0	4370	1.9	7200
16	20	0	301	0	390
	40	0.1	1411	0.1	2897
	100	2.9	7200	0.8	7200
	150	4.8	7200	2.7	7200

Table 5.1: Computational Results of the Branch-and-Price Approach on Small Instances

Table 5.1 also reveals that performance of the branch-and-price approach does not differ considerably when the ratio of the over-pivot weight to the pivot weight varies. With the remaining parameters being the same, 8 cases with $\frac{U_j^E}{U_j} = 10\%$ and 8 cases with $\frac{U_j^E}{U_j} = 30\%$ are faster or achieve a lower gap.

5.6.2 Performance of the Best-First Decreasing Heuristic

Tables 5.2 and 5.3 present a comparison between the branch-and-price approach and the best-fit decreasing heuristic. Their performance is measured against running CPLEX for 7200 seconds. The "Diff(%)" column calculates the percentage difference between the best feasible solution of the algorithm and CPLEX, where $Diff(\%) = \frac{f_{method} - f_{cplex}}{f_{cplex}} \times 100\%$. The "gap(%)" column displays the gap relative to CPLEX lower bound, i.e. $gap(\%) = \frac{f_{method} - LB}{LB}$. Solution times for BFDH and branch-and-price are listed side-by-side with the CPLEX execution time. The 7200 in "CPLEX Time" column implies that CPLEX reaches the time threshold before reaching optimum. The "*" indicates that CPLEX is running out of memory before reaching the time threshold. The total-node-size limit is set to 2GB for the CPLEX solver. The default tolerance gap for CPLEX is set to 0.001%.

Although the branch-and-price algorithm performed well for small instances, it cannot find the optimal solution within the 7200-second threshold when the numbers of shipments and ULDs increase. We resort to the BFDH of Section 5.4 for large instances. BFDH takes a relatively short time for large instances while maintaining a relatively smaller gap compared to CPLEX's best feasible solution. Branch-and-price delivers better solution quality, but with longer computational time.

$\lvert J\rvert$	N	$U_j^E = 10\%U_j$							$U_j^E = 30\%U_j$						
		BFDH Dif-f(%)	BP D-iff(%)	BFDH gap(%)	BP gap(%)	BFDH Time	BP Time	CPLEX Time	BFDH Dif-f(%)	BP D-iff(%)	BFDH gap(%)	BP gap(%)	BFDH Time	BP Time	CPLEX Time
8	20	0.0	0.0	0.0	0.0	11	205	10	0.0	0.0	0.0	0.0	13	179	8
	40	0.3	0.0	0.3	0.0	104	426	27	0.0	0.0	0.0	0.0	145	802	20
	100	0.1	0.0	0.1	0.0	259	1703	213	0.0	0.0	0.0	0.0	292	780	505
	150	2.8	0.0	2.8	0.0	269	3077	1558	1.5	0.0	1.5	0.0	310	6023	1497
16	20	0.0	0.0	0.0	0.0	170	301	24	0.0	0.0	0.0	0.0	189	390	17
	40	0.0	0.0	0.0	0.0	295	1411	35	0.0	0.0	0.0	0.0	180	2897	41
	100	1.6	2.2	1.6	2.2	382	7200	538	0.8	3.1	0.8	3.1	354	7200	343
	150	2.5	0.0	2.5	0.0	379	7200	3203	2.6	1.9	2.6	1.9	418	7200	2940
40	100	1.5	0.0	1.5	0.0	290	4159	3825	2.0	0.0	2.0	0.0	303	2619	1564
	200	3.1	0.0	3.1	0.0	212	7200	7200	2.8	0.0	2.8	0.0	286	3951	2974
	300	4.9	1.8	6.2	3.1	265	7200	*	2.1	2.3	3.2	3.4	199	7200	*
	400	2.8	0.3	5.8	3.3	381	7200	*	5.8	3.2	6.3	3.7	421	7200	*
80	100	2.0	0.0	2.0	0.0	99	3574	7200	3.9	0.0	3.9	0.0	157	955	820
	200	3.5	0.0	3.5	0.0	201	7200	7200	2.5	1.3	2.5	1.3	168	7200	2463
	300	3.6	0.0	4.6	0.9	328	7200	*	4.4	2.6	4.4	2.6	410	7200	4982
	400	2.3	3.1	3.6	4.4	498	7200	*	2.8	2.4	4.4	4.0	407	7200	*

Table 5.2: Comparison between Branch-and-Price and BFDH for $c_j^E/c_j = 1.2$

The results from Tables 5.2 and 5.3 also reveal that CPLEX takes slightly shorter time (13% on average) for cases with higher over-pivot cost ($c_j^E/c_j = 3$). There is no significant difference in the branch-and-price solution quality for cases with different c_j^E/c_j ratios, although its mean execution time is 5% less than that of CPLEX. For BFDH, the typical execution time is shorter (9.6% shorter on average) when $c_j^E/c_j = 3$, and its mean difference from CPLEX is also smaller. In the airfreight industry, the c_j^E/c_j ratio is usually set between 1.1 and 2. This ratio increases significantly only during peak seasons for certain tradelanes.

5.6.3 Performance of The Two Local Branching Extensions

Tables 5.4 and 5.5 report the performance of the two extension of local branching algorithms given in Sections 5.5.2 and 5.5.3. We use the same instances that we did for branch-and-price and BFDH (as in Tables 5.2 and 5.3). We list the performance of local branching with multiple trees (MultiTrees) and the relaxation-induced search (RINS) in terms of percentage difference from the best solution that CPLEX achieves after 2 hours ("Diff(%)"). The negative percentage figures imply that the method finds a better solution than CPLEX. Computational results of the two local branching extensions are also compared with the CPLEX lower bound (gap(%)).

Relative to results in Table 5.2, RINS achieves better solution quality than CPLEX, BFDH and Branch-and-Price for cases with more than 40 ULDs and 300 shipments. With respect to solution speed alone, the best-first decreasing heuristic can finish all computations within 500 seconds. But its superior solution speed is compromised by its

$\lvert J\rvert$	N	$U_j^E = 10\%U_j$							$U_j^E = 30\%U_j$						
		BFDH Dif-f(%)	BP D-iff(%)	BFDH gap(%)	BP gap(%)	BFDH Time	BP Time	CPLEX Time	BFDH Dif-f(%)	BP D-iff(%)	BFDH gap(%)	BP gap(%)	BFDH Time	BP Time	CPLEX Time
8	20	0.0	0.0	0.0	0.0	15	91	15	0.0	0.0	0.0	0.0	12	105	20
	40	0.0	0.0	0.0	0.0	99	381	13	0.1	0.0	0.1	0.0	133	604	31
	100	0.7	0.0	0.7	0.0	198	1558	613	1.8	0.0	1.8	0.0	346	901	238
	150	2.0	0.0	2.0	0.0	310	3384	596	2.9	2.1	2.9	2.1	339	3965	586
16	20	0.0	0.0	0.0	0.0	86	245	25	0.0	0.0	0.0	0.0	176	294	6
	40	0.2	0.0	0.2	0.0	267	2893	9	2.5	0.0	2.5	0.0	190	3280	75
	100	0.0	0.0	0.0	0.0	342	1387	378	2.6	0.0	2.6	0.0	218	5085	504
	150	1.9	0.0	1.9	0.0	415	6282	2899	1.4	0.0	1.4	0.0	277	4237	471
40	100	2.4	0.0	2.4	0.0	248	3354	1228	1.7	0.0	1.7	0.0	262	3874	401
	200	2.1	0.2	2.1	0.2	430	7200	7200	5.1	0.3	5.1	0.3	175	7200	2089
	300	4.8	0.0	4.8	0.0	305	3752	2017	0.8	0.2	0.8	0.2	463	7200	4995
	400	2.0	0.0	3.0	1.0	361	7200	*	4.6	1.1	4.6	1.1	408	7200	4681
80	100	3.6	0.0	3.6	0.0	184	1103	7200	4.0	0.0	4.0	0.0	204	3891	3325
	200	6.1	1.8	6.1	1.8	223	7200	5339	1.3	0.0	2.1	0.8	191	7200	*
	300	2.7	1.6	2.7	1.6	426	7200	2621	2.8	2.5	2.8	2.5	373	7200	3893
	400	4.7	0.0	4.7	0.0	393	5781	7200	4.9	2.8	4.9	2.8	487	7200	2128

Table 5.3: Comparison between Branch-and-Price and BFDH for $c_j^E/c_j = 3$

$\|J\|$	N	$U_j^E = 10\%U_j$					$U_j^E = 30\%U_j$				
		Diff (%) Multi Trees (20min)	Diff (%) RINS (20min)	Multi Trees gap(%)	RINS gap(%)	CPLEX Time (2hrs)	Diff (%) Multi Trees (20min)	Diff (%) RINS (20min)	Multi Trees gap(%)	RINS gap(%)	CPLEX Time (2hrs)
8	20	0.0	0.0	0.0	0.0	10	0.0	0.0	0.0	0.0	8
	40	0.3	0.0	0.3	0.0	27	0.2	0.0	0.2	0.0	20
	100	1.6	2.9	1.6	2.9	213	2.6	0.4	2.6	0.4	505
	150	2.7	0.5	2.7	0.5	1558	3.3	0.4	3.3	0.4	1497
16	20	1.2	0.0	1.2	0.0	24	0.9	0.0	0.9	0.0	17
	40	0.8	0.8	0.8	0.8	35	0.0	0.0	0.0	0.0	41
	100	0.0	0.9	0.0	0.9	538	2.0	0.1	2.0	0.1	343
	150	1.2	3.4	1.2	3.4	3203	0.9	0.9	0.9	0.9	2940
40	100	1.9	0.4	1.9	0.4	3825	2.8	0.6	2.8	0.6	1564
	200	1.5	0.8	1.5	0.8	7200	1.9	1.4	1.9	1.4	2974
	300	1.8	-1.2	3.1	0.1	*	1.9	-0.8	3.1	0.3	*
	400	0.6	-2.7	3.6	0.3	*	3.5	-0.4	4.0	0.1	*
80	100	2.1	0.6	2.1	0.6	7200	4.1	1.1	4.1	1.1	820
	200	3.4	1.9	3.4	1.9	7200	4.8	2.3	4.8	2.3	2463
	300	-0.5	-0.5	0.4	0.4	*	1.1	1.1	1.1	1.1	4982
	400	-0.4	-0.9	0.9	0.4	*	2.0	-0.9	3.6	0.7	*

Table 5.4: Comparison of Multi-Trees and RINS Approaches for $c_j^E/c_j = 1.2$

$\lvert J\rvert$	N	$U_j^E = 10\%U_j$					$U_j^E = 30\%U_j$				
		Diff(%) Multi Trees (20min)	Diff(%) RINS (20min)	Multi Trees gap(%)	RINS gap(%)	CPLEX Time (2hrs)	Diff(%) Multi Trees (20min)	Diff(%) RINS (20min)	Multi Trees gap(%)	RINS gap(%)	CPLEX Time (2hrs)
8	20	0.0	0.0	0.0	0.0	15	0.0	0.0	0.0	0.0	20
	40	0.0	0.0	0.0	0.0	13	0.0	0.0	0.0	0.0	31
	100	0.0	0.0	0.0	0.0	613	0.1	0.0	0.1	0.0	238
	150	0.5	0.2	0.5	0.2	596	0.4	0.4	0.4	0.4	586
16	20	0.0	0.0	0.0	0.0	25	0.2	0.0	0.2	0.0	6
	40	0.0	1.8	0.0	1.8	9	2.2	0.4	2.2	0.4	75
	100	0.8	2.9	0.8	2.9	378	0.3	1.2	0.3	1.2	504
	150	2.4	1.0	2.4	1.0	2899	1.3	2.3	1.3	2.3	471
40	100	2.1	0.0	2.1	0.0	1228	1.9	1.1	1.9	1.1	401
	200	1.2	1.2	1.2	1.2	7200	1.3	0.6	1.3	0.6	2089
	300	6.1	0.7	6.1	0.7	2017	3.4	0.5	3.4	0.5	4995
	400	2.1	-0.8	3.1	0.2	*	2.1	0.3	2.1	0.3	4681
80	100	2.4	0.6	2.4	0.6	7200	1.2	1.2	1.2	1.2	3325
	200	4.0	1.5	4.0	1.5	5339	2.5	-0.5	3.3	0.3	*
	300	3.9	0.0	3.9	0.0	2621	0.0	3.2	0.0	3.2	3893
	400	1.9	1.1	1.9	1.1	7200	1.6	1.6	1.6	1.6	2128

Table 5.5: Comparison of Multi-Trees and RINS Approaches for $c_j^E/c_j = 3$

solution quality, with an average difference of 3.6% from CPLEX for cases with $n \geq 40$. Therefore, it is advised to use branch-and-price if solution quality is pursued, and BFDH if a fast solution is desired. In contrast, the two local branching extensions provide relatively good balance between solution quality and speed.

Table 5.4 reveals that running the RINS algorithm in 20 minutes outperforms CPLEX's best solution in 7 out of 16 large cases ($n \geq 40$) when $U_j^E = 10\% U_j$, and for two cases when $U_j^E = 30\% U_j$. RINS also outperforms the multi-local-tree approach in all but two cases when $n \geq 40$. However, when the number of ULDs is low ($n \leq 16$), the two local branching extensions do not outperform CPLEX. When the ratio of c_j^E/c_j increases to 3.0 in Table 5.5, RINS finds a better solution than CPLEX in only two cases, while the multi-local-tree approach does not give better solutions for *any* of the cases.

Although RINS outperforms multi-tree when the number of ULDs is large, multi-tree provides better or equal results compared to RINS in 19 out of 32 cases for $n \leq 16$. This is because, when the ratio between the number of shipments and number of ULDs is large, creating parallel local trees adds extra redundancy to the computation. This also reminds us to create multiple local trees *dynamically*, according to the ratio of the first-level and second-level branching variables. When $U_j^E = 30\% U_j$, there is no substantial difference in performance compared to when the over-pivot capacity is 10% of the pivot weight. The gap is relatively small for $c_j^E/c_j = 3$ compared to that when $c_j^E/c_j = 1.2$.

In the analysis above, both the multi-tree and the RINS are based on the multi-level-variable approach. In Table 5.6, we compare different combination of the local branching heuristics, with or without the multi-level variables. For each combination (n, m), we generated four instances and took the average. The ratio of c_j^E/c_j was fixed at 1.2 for all cases. The time limit for local branching operation was set to 20 minutes. Comparison was made to CPLEX's lower bound after 7200 seconds.

$\|J\|$	N	MultiLevel Gap(%)	MultiTrees Gap(%)	RINS Gap(%)	MultiLevel & MultiTrees Gap(%)	MultiLevel & RINS Gap(%)
10	20	0.2	0.3	0.1	0.0	0.0
	100	4.9	1.3	8.4	0.0	0.1
	200	4.8	4.0	5.8	1.1	0.7
	400	6.8	3.9	3.9	4.8	1.9
20	20	0.2	2.2	0.5	0.3	0.0
	100	2.8	3.1	2.9	1.3	0.0
	200	2.9	4.9	3.5	2.2	0.0
	400	8.1	3.7	4.2	3.0	1.4
40	20	3.9	3.5	2.0	0.0	0.2
	100	3.6	2.8	2.1	1.2	1.8
	200	7.0	4.0	3.8	3.3	1.4
	400	9.1	3.4	4.5	2.6	1.1
80	20	5.9	1.8	0.9	1.2	0.5
	100	5.0	4.1	6.1	1.0	1.9
	200	6.4	3.3	6.7	2.5	2.0
	400	7.7	3.5	4.9	1.9	1.3
Average		5.0	3.1	3.8	1.7	0.9

Table 5.6: Average Performance Comparison of Different Local Branching Extensions

Table 5.6 reveals that a combination of multi-level-variable and relaxation-induced neighborhood search achieves better quality solution (on average) than other combinations. This combination gives a mean difference of 0.9% from the CPLEX lower bound. Using multi-local trees alone, variances are small among instances of different sizes; however, the overall algorithm is slow to run, and it explores less of the solution space in a given time threshold. If we directly run the multi-local-tree approach of Section 5.5.2 (without taking multi-level-variable into account), the diversification is less effective for z_j. Low-cost ULDs should always be chosen greedily. However, we need to run more local trees at the branching of x_{ij}, as the algorithm tends to get stuck at local optima at that level. A two-level approach with a diversification focus on x_{ij} is more effective, as seen when comparing the columns "MultiLevel" and "MultiLevel & MultiTrees". Performance of RINS is poor when we don't consider two levels of binary variables. This is because, when z_j has the same integer value in the incumbent solution and in the LP relaxation, that is likely to be the optimal value; the same conclusion seldom holds for x_{ij}.

5.6.4 Multiple ULD Classes

We also looked at problem with multiple classes of ULDs. In the following numerical cases, each test case is characterized by three parameters: (i) the number of ULD types available: m (ii) the number of shipments: N (iii) the number of ULDs per type: q. For every N shipments, with the total ULD capacity remaining the same (i.e. mq=constant), we have three values for the number of ULD types. Differences between those types are always expressed in terms of ULD capacities U_j.

Table 5.7 reports the computational performance of our local branching (with RINS) and that of CPLEX in two hours. When the cost c_j^E is closer to c_j, there is a greater mean difference of local branching with RINS than for cases with high c_j^E/c_j ratio. When the total number of shipments is small, our algorithm performs better when there are fewer

N	m	q	$c_j^E/c_j = 1.2$			$c_j^E/c_j = 3$		
			RINS (15 min)	RINS (30 min)	CPLEX Time	RINS (15 min)	RINS (30 min)	CPLEX Time
40	2	15	0.0	0.0	445	0.0	0.0	294
	3	10	1.2	0.2	270	0.3	0.3	233
	5	6	0.1	0.1	289	2.7	0.0	108
80	2	15	0.3	0.0	1887	2.4	0.0	925
	3	10	1.1	0.8	1284	1.4	0.0	898
	5	6	2.2	2.0	432	3.9	0.5	379
200	2	15	1.5	0.0	7200	0.8	-0.3	7200
	3	10	1.4	0.8	6224	2.6	1.1	4045
	5	6	0.9	-0.2	7200	0.8	0.8	3437
400	2	15	0.1	-0.1	7200	0.1	-0.3	7200
	3	10	1.6	0.4	7200	1.0	0.0	1081
	5	6	0.0	-1.0	7200	1.1	-0.1	7200

Table 5.7: Local Branching with Different Classes of ULDs

classes of ULDs. However, when the number of shipments goes up, our algorithm performs better than CPLEX when there is a larger number of ULD classes.

5.7 Summary

In this chapter, we proposed three methodologies to solve a pivot-weight based air cargo load-planning problem. The first one is branch-and-price that is computationally slow. The second is a best-first decreasing heuristic that is fast and have an acceptable solution quality.

Finally, in addition to the multi-level-variables extension we raised in Chapters 3 and 4, we extended the local branching heuristic to include multiple local trees and to use a relaxation-induced neighborhood approach. The latter extension proved effective in leading to high quality solutions in reasonable computational time. Problems with up to 400 shipments and 80 containers are proved to be solved to within 3.4% of optimality in less than 20 minutes.

Chapter 6

Conclusions and Future Research Directions

In this dissertation, we have discussed three problems that arise in the airfreight industry and presented several solution methodologies to solve them.

In Chapter 3, we proposed a network shipment consolidation model that features the unsplittable shipment requirement, volume weight, bumping clause, multiple origins and flight/shipment time. We introduced a decomposition approach to solve the model. Upon a Lagrangian relaxation, this problem is decomposed to two sets of well-known problems: a minimum cost flow problem and a knapsack problem. For larger problem sizes, we relied on subgradient optimization and local branching. We also applied a multi-level-variable local branching strategy, which delivered superior computational performance. Local branching delivered the best solution for the tightly-capacitated cases, while subgradient optimization gave the better solution for loosely-capacitated cases. From the operations manager's perspective, the local branching algorithm proved to be most effective when total customer demand is close to the capacity reserved from airlines. In those cases, our algorithm achieves much faster and more accurate solutions than an MIP commercial solver such as CPLEX.

In Chapter 4, we added more practical features to an existing network flow problem with cross-arc costs. This resulted in a model for airfreight forwarders that helps to solve the consolidation problem under a *system-wide* discount. Moreover, we extended our model such that it is capable of solving scenarios when both a flight-leg and a system-wide discount are offered to a freight forwarder. As practical instances of daily operation under such a combination of discounts are of huge size, we developed a local branching heuristic that is capable of solving such problems in a relatively short time. Branch-and-price delivered only poor results within half an hour, for the larger cases; local branching beat branch-and-price in both solution speed and quality. From the managerial perspective, the availability of the double-discount reduces the average number of flights to which a freight forwarder tender needs to tender loads. As a result of the double discount, the forwarder achieves a subsequent saving on operational and administrative cost.

For the ACPW problem in Chapter 5, the decisions are focused on shipments to be consolidated into ULD, as opposed to general cargo (goods that are not containerized) in Chapters 3 and 4. We proposed a decomposition strategy that separates the problem into each ULD. The branch-and-price approach is able to produce an exact solution for smaller problem sizes. However, the multi-level-variable local branching approach we employed for the first two topics does not work well on the ACPW. Instead, we proposed multiple local trees and a relaxation-induced neighborhood search (mixed variable fixing) as extensions to the basic local branching framework. We found that the relaxation-induced neighborhood search yielded the best computational performance. From the managerial perspective, the model we proposed for the ACPW problem satisfies an important need for airfreight forwarders, since air cargo is containerized to a greater degree nowadays. Our BFDH and RINS algorithm make it possible for an operations manager to obtain close-to-optimal decision, for an entire day, within 15 minutes.

We recommend the following for future research. Our focus on the pivot-weight problem has so far been restricted to gross weight and not volume weight. Nor we distinguish between cargo that is loaded in the upper deck vs lower deck. It would be very promising if this problem could be extended to either or both of these settings. Moreover, this problem has the potential to be combined with a shipment-routing problem. We hope our work on the pivot-weight problem can be integrated with other traditional transportation and logistics models.

Moreover, our local branching approach can be used to efficiently handle constraints such as incompatibility of items that should not be loaded together: Constraints such as $x_1 + ... + x_n \leq l$ can be included in an adaptive local branching approach. In fact, any consolidation problem with disjunctive constraints, or with constraints that can be expressed as a linear combination of other disjunctive constraints, is able to be processed efficiently using local branching. Future research can focus on variations of our problems with some practical disjunctive constraints.

Aside from airfreight rules such as volume weight, all-unit discount and pivot weight, another common rule in air transportation is the minimum-weight restrictions on ULDs. This resembles a special case of the bin packing problem with minimum filling constraint (BPPMFC), where each container should be loaded to more than a minimum threshold, or proportional to its volume, to guarantee flight safety.

In this thesis, we have focused on the forwarder's cost-saving decisions from an operational point of view. Another stream of revenue- and cost-saving opportunities comes from accurate *forecasting*. Instead of treating the amount of reserved capacity as constant, freight forwarders make reservation decisions at different stages of planning. In reality, some demands are placed just eight hours before flight take-off. Hence, not all shipments are known when the flight bookings are made by forwarders. Effectively dealing with this dynamic nature is critical to the success of planning. Moreover, there are generally two rounds of booking for airfreight forwarders. In Round 1, six to twelve months before the actual departure, freight forwarders bid for cargo space. In Round 2, a few days before the actual take-off, the freight forwarders have to confirm the allotted space, either returning unwanted space or confirming their need for the whole allotted capacity. The remaining capacity is available for free sale. There are two aspects we can work on in this case: First, we could consider the booking capacity on each flight to be a decision variable with unknown shipment information. Second, we could still make shipment routing decisions, but with shipment information unknown.

In this thesis, all models are derived from the freight forwarder's perspective. In fact, cargo airlines also have their core decision-making mechanism to maximize their own operating profit. From the air-carrier's perspective, there is uncertainty in the cargo that will be tendered. Until departure, the airlines do not know how much capacity they have available for free sale. To ensure space on constrained flights, freight forwarders intentionally bid on more capacity than they actually need, since most airlines allow the return of unwanted space at little or no extra charge. The airlines add the released space to the pool of capacity available for free sale. In addition, for planes carrying both cargo and passengers (combination carriers), the cargo space usually contains passengers' baggage and cargo in the same compartment. These factors, plus weather (which affects the amount of fuel on board the aircraft) and mail, influence how much capacity is actually available for free sale. Finally, cargo space is constrained by two dimensions, weight and volume. Prior to departure, however, the airline typically does not know which will be the most restrictive.

Each of the preceding possibilities suggests that research on these topic extensions will remain exciting for some time to come.

References

H. Akeb, M. Hifi, and M. E. O. A. Mounir. Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60 (4):811 – 820, 2011.

C. Alves and J. V. de Carvalho. Accelerating column generation for variable sized bin-packing problems. *European Journal of Operational Research*, 183(3):1333 – 1352, 2007.

A. Amiri and H. Pirkul. New formulation and relaxation to solve a concave-cost network flow problem. *The Journal of the Operational Research Society*, 48(3):278–287, 1997.

J. S. Ang, C. Cao, and H.-Q. Ye. Model and algorithms for multi-period sea cargo mix problem. *European Journal of Operational Research*, 180(3):1381 – 1393, 2007.

A. P. Armacost, C. Barnhart, K. A. Ware, and A. M. Wilson. UPS optimizes its air network. *Interfaces*, 34(1):15–25, 2004.

A. Atamtürk and D. Rajan. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Mathematical Programming*, 92(2):315–333, 2002.

A. Balakrishnan and S. C. Graves. A composite algorithm for a concave-cost network flow problem. *Networks*, 19(2):175–202, 1989.

C. Barnhart and S. Shen. Logistics service network design for time-critical delivery. *Practice and Theory of Automated Timetabling V Lecture Notes in Computer Science*, 3616:86–105, 2005.

C. Barnhart, C. A. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

C. Barnhart, N. Krishnan, D. Kim, and K. Ware. Network design for express shipment delivery. *Computational Optimization and Applications*, 21:239–262, 2002.

E. Bartolini and A. Mingozzi. Algorithms for the non-bifurcated network design problem. *Journal of Heuristics*, 15(3):259–281, 2009.

M. Belaidouni and W. Ben-Ameur. On the minimum cost multiple-source unsplittable flow problem. *RAIRO-Operations Research*, 41(3):253–273, 2007.

J. H. Bookbinder and J. K. Higginson. Probabilistic modeling of freight consolidation by private carriage. *Transportation Research Part E*, 38:305–318, 2002.

G. Brønmo, B. Nygreen, and J. Lysgaard. Column generation approaches to ship scheduling with flexible cargo sizes. *European Journal of Operational Research*, 200(1):139 – 150, 2010.

J. Carter, B. Ferrin, and C. Carter. The effect of less-than-truckload rates on the purchase order lot size decision. *Transportation Journal*, 34(3):35–44, 1995.

A. Chabrier. Heuristic branch-and-price-and-cut to solve a network design problem. In *Proceedings of CPAIOR'03*, Madrid, 2003.

T.-S. Chang. Best routes selection in international intermodal networks. *Computers & Operations Research*, 35(9):2877 – 2891, 2008.

N. Cherfi and M. Hifi. A column generation method for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, 46:51–73, 2010.

A. Cohn, M. Davey, L. Schkade, A. Siegel, and C. Wong. Network design and flow problems with cross-arc costs. *European Journal of Operational Research*, 189(3):890 – 901, 2008.

T. G. Crainic, M. Gendreau, and J. M. Farvolden. A simplex-based tabu search method for capacitated network design. *Informs Journal on Computing*, 12(3):223–236, 2000.

T. G. Crainic, G. Perboli, W. Rei, and R. Tadei. Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers & Operations Research*, 38 (11):1474–1482, 2011.

K. L. Croxton, B. Gendron, and T. L. Magnanti. Models and methods for merge-in-transit operations. *Transportation Science*, 37(1):1–22, 2003.

E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.

G. B. Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28(1):30–44, 1960.

Z. Degraeve and M. Peeters. Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *Informs Journal on Computing*, 15(1):58–81, 2003.

S. Elhedhli, L. Li, M. Gzara, and J. Naoum-Sawaya. A branch-and-price algorithm for the bin packing problem with conflicts. *Informs Journal on Computing*, 23(3):404–415, 2011.

M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

M. Fischetti, C. Polo, and M. Scantamburlo. A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44(2):61–72, 2004.

M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

A. Frangioni. About lagrangian methods in integer optimization. *Annals of Operations Research*, 139:163–193, 2005.

I. Ghamlouche, T. G. Crainic, and M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51(4):655–667, 2003.

G. Hadley. *Non-Linear and Dynamic Programming.* Addison-Wesley, New York, 1964.

R. W. Hall. Consolidation strategy: Inventory, vehicles and terminals. *Journal of Business Logistics*, 8:57–73, 1987.

P. Hansen, N. Mladenovic, and D. Urosevic. Variable neighborhood search and local branching. *Computers & Operations Research*, 33(10):3034 – 3045, 2006.

J. K. Higginson and J. H. Bookbinder. Policy recommendations for a shipment-consolidation program. *Journal of Business Logistics*, 15(1):87–112, 1994.

K. Huang and W. Chi. A lagrangian relaxation based heuristic for the consolidation problem of airfreight forwarders. *Transportation Research Part C*, 15(4):235 – 245, 2007.

A. H. G. R. Kan, L. Stougie, and C. Vercellis. A class of generalized greedy algorithms for the multi-knapsack problem. *Discrete Applied Mathematics*, 42(2-3):279 – 290, 1993.

J. Leung, M. Dror, and G. Young. A note on an open-end bin packing problem. *Journal of Scheduling*, 4(4):201–207, 2001.

L. C. Leung, Y. Van Hui, Y. Wang, and G. Chen. A 0-1 LP Model for the Integration and Consolidation of Air Cargo Shipments. *Operations Research*, 57(2):402–412, 2009.

Y. Li, Y. Tao, and F. Wang. A compromised large-scale neighborhood search heuristic for capacitated air cargo loading planning. *European Journal of Operational Research*, 199 (2):553 – 560, 2009.

Z. Li, J. H. Bookbinder, and S. Elhedhli. Optimal shipment decisions for an airfreight forwarder: Formulation and solution methods. *Transportation Research Part C*, 21(1): 17 – 30, 2012.

D. Lichtenberger. *An Extended Local Branching Framework and its Application to the Multidimensional Knapsack Problem.* PhD thesis, der Technischen Universitat Wien, 2005.

R. K. Martin. *Large Scale Linear and Integer Optimization : A Unified Approach.* Kluwer Academic, Boston, 1999.

A. Muriel and F. Munshi. Capacitated multicommodity network flow problems with piecewise linear concave costs. *IIE Transactions*, 36(7):683–696, 2004.

G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382 – 392, 2002.

J. Puchinger, G. R. Raidl, and S. Pirkwieser. Metaboosting: Enhancing integer programming techniques by metaheuristics. In V. Maniezzo, T. Sttzle, and S. Vo, editors, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 71–102. Springer, 2010.

I. Rodriguez-Martin and J. J. Salazar-Gonzalez. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37(3):575 – 581, 2010.

S. Root and A. Cohn. A novel modeling approach for express package carrier planning. *Naval Research Logistics*, 55(7):670–683, 2008.

L. Schenk and D. Klabjan. Intramarket optimization for express package carriers. *Transportation Science*, 42(4):530–545, 2008.

P. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9:211–228, 1998.

C. Wallace. *Mixed integer programming heuristics*. PhD thesis, Carnegie Mellon University, 2010.

W. H. Wong, L. C. Leung, and Y. V. Hui. Airfreight forwarder shipment planning: A mixed 0-1 model and managerial issues in the integration and consolidation of shipments. *European Journal of Operational Research*, 193(1):86–97, 2009.