OPERATIONS SCHEDULING WITH DELIVERY DEADLINES IN MULTI-ECHELON SUPPLY CHAINS

By

GANG WANG

A Dissertation submitted to the

Graduate School - Newark

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Management Science

written under the direction of

Dr. Lei Lei

and approved by

____

____

____

____

____

____

Newark, New Jersey

October, 2012

ABSTRACT OF THE DISSERTATION

Operations Scheduling with Delivery Deadlines in Multi-Echelon Supply Chains

by GANG WANG

Dissertation Director: Dr. Lei Lei

We study the operations scheduling problem with delivery deadlines over capacitated multi-echelon shipping networks. Our main results consist of new mathematical models, structural analysis, and solution methodologies for this type of operations scheduling problems, which are in general computationally difficult due to their inherent combinatorial nature.

Part I of the dissertation investigates three polynomial-time solvable cases, including case **1)** identical order sizes; case **2)** designated suppliers; and case **3)** divisible order sizes. For the first case, we prove that the original problem can be decomposed into two sub-problems: the transportation problem and a specially structured mixed integer programming model that is totally unimodular. For the second case, we show that the original problem can be solved by the Minimal Spanning Tree algorithm that runs in polynomial time. The third case is shown to be solvable in polynomial time by extending the literature results for a special case of the well- known bin packing problem. Part II of this dissertation analyzes the structure properties of the network scheduling problem with a single processing center (PC) between the suppliers and customers. A dynamic programming-based search algorithm that correctly identifies the optimal subset of customer orders to be fulfilled under each given utilization level of the PC capacity is proposed. We also prove that the resulting search algorithm converges to the optimal solution within pseudo-polynomial time. Part III of the dissertation focuses on the methodology of solving the general operations scheduling problems with customer delivery deadlines. We propose a linear programming relaxation-based algorithm. With this algorithm, a given network scheduling problem is solved through an iterative process. During each iteration, a threshold parameter is used to select the relaxed linear variables to be binary variables for the next iteration, while a subset of binary variables is still relaxed to bounded linear variables. The iteration continues until the values of all the binary variables are determined. This partial relaxation allows us to avoid dealing with the generalized knapsack problem, a difficult NP-hard problem, in the solution process.

Preface

This Ph.D. thesis contains the results of the research carried out at the Department of Supply Chain Management and Marketing Sciences, Rutgers Business School, Rutgers University from January 2008 to July 2012, entitled "Operations Scheduling with Delivery Deadlines in Multi-Echelon Supply Chains."

The subject of this Ph.D. thesis is on new methodologies for solving the operations scheduling problem with delivery deadlines on the capacitated multi-echelon shipping networks. We develop mathematical models to describe the multi-echelon supply chain operations subject to network capacity and delivery timeliness constraints. Furthermore, we analyze some special cases related to the operations scheduling problem, thus propose new solution approaches, and report the effectiveness of these methodologies based on the numerical or computational experiments. The main contribution of this thesis is that our research results extend and improve the existing approaches in the literature for solving the operations scheduling problems with delivery deadlines in multi-echelon supply chain.

The thesis is submitted as a partial fulfillment of the requirement for obtaining the Ph.D. degree at the Rutgers University. The project was supported by the Dissertation Fellowship of the Graduate School, Rutgers University.

Acknowledgements

Five years has been a challenging trip, with both ups and downs. It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Above all, I would like to thank my advisor, Prof. Lei Lei, for her thoughtful guidance and great patience at all times, not to mention her invaluable advice and unsurpassed knowledge of Supply Chain Management. Moreover, my career path would not be clear as it is now without her inspiration, help and support on both academic and personal level, for which I extremely appreciate.

I would also like to express sincere appreciation to my committee members: Prof. Yao Zhao, Prof. Tan C. Miller at Rider University, and Prof. Yeinyurt for their valuable time, efforts, and insightful comments on the drafts of my dissertation. I would like to acknowledge the professors and the staff at the Department of Supply Chain Management and Marketing Sciences for their support and assistance during my wonderful five years at Rutgers Business School.

I would also like to thank my colleagues and friends at Rutgers Business School for their comfort and encouragement. Last, but by no means least, I would like to thank my mother and younger uncle, who have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

Gang Wang

New Jersey, July 2012

Table of Contents

List of Tables

List of Figures

List of Symbols

S	Set of contracted suppliers, i.e., $\mathrm{S}=\{1,2,\cdots,S\}$
J	Set of demand points, i.e., $\mathrm{J}=\{1,2,\cdots,J\}$
N	Set of processing centers or PCs, $\mathrm{N}=\{1,2,\cdots,N\}$
F_s	Capacity of supplier s , $s \in \mathrm{S}$
b_{sn}	Shipping rate (\$/unit) from supplier s to $\mathrm{PC_n}$, $s \in \mathrm{S}$, $n \in \mathrm{N}$
t_{sn}	Shipping time from supplier s to $\mathrm{PC_n}$, $s \in \mathrm{S}$, $n \in \mathrm{N}$
C_n	Processing capacity of $\mathrm{PC_n}$, $n \in \mathrm{N}$
τ_n	Unite processing time at $\mathrm{PC_n}$, $n \in \mathrm{N}$
λ_j	Order quantity of demand point j , $j \in \mathrm{J}$
a_{nj}	Shipping rate from $\mathrm{PC_n}$ to demand point j , $n \in \mathrm{N}$, $j \in \mathrm{J}$
π_{nj}	Fixed shipment cost from $\mathrm{PC_n}$ to demand point j , $n \in \mathrm{N}$, $j \in \mathrm{J}$
t_{nj}	Shipping time from $\mathrm{PC_n}$ to demand point j , $n \in \mathrm{N}$, $j \in \mathrm{J}$
p_j	Unit penalty cost for unsatisfied demand point j , $j \in \mathrm{J}$
T_j	Deadline of delivering the order to demand point j , $j \in \mathrm{J}$
m	The total number of distinct customer orders
d_i	i th order size, $i=1,2,\cdots,m$, and $d_1 < d_2 < \cdots < d_m$
Φ_i	The subset of orders of size d_i , $1 \le i \le m$. Let $n_i = \lvert\Phi_i\rvert$
r_n^1	The residual capacity of $\mathrm{PC_n}$ only used for orders of size d_1, $r_n^1 = (C_n \bmod d_2)$, $\forall n \in \mathrm{N}$
h_1	The maximum number of fulfilled orders of size d_1 by utilizing the PC residual capacity r_n , $\forall n \in \mathrm{N}$, and $h_1 = \min\left\{n_1, \sum_{n=1}^{N} \left\lfloor r_n^1 / d_1 \right\rfloor\right\}$

1 Introduction

Increasing global operations, greater cost pressure, and more volatile market dynamics are making the supply chain more complex and thus creating an ever-growing demand for effective and efficient supply chain management. According to a survey from PRTM, most applicants expect that future business growth will come primarily from international customers and customized products, and more than 85% of companies expect their supply chain complexity to increase significantly (PRTM Management Consultants, 2011). In addition, a survey by Aberdeen Group revealed that the impact of increasing supply chain complexity (i.e., longer lead times, increasing number of suppliers, partners, carriers, customers, countries, logistics channels) and rising supply chain management costs (e.g., total landed costs, fuel costs, labor costs) are the current top business pressures (Aberdeen Group, 2011). To succeed in the global marketplace during the economic upturn, companies must capitalize on the compelling international market's opportunities and prepare their supply chain operations for the supply chain challenges. However, despite strong growing expectations on effectively managing the supply chain operations, the findings by PRTM indicated that many companies lacked the capabilities critical for meeting growing demand or for managing an increasingly complex and global supply chain. The survey also showed that approximately 30% mentioned the lack of integration between supply chain functions. Integrated supply chain management across all key functions still seems to be a myth, with many procurement and manufacturing vice presidents making soloed optimization decisions.

Therefore, one of the many optimization problems to be solved regarding supply chain performance optimization is the operations scheduling problem over multi-echelon shipping networks under the consideration of timely delivery deadlines and customized products, i.e., integrated supply and distribution problem.

1.1 Motivation

Our study on the supply chain operations scheduling problem of the multi-echelon shipping network was motivated by the practice of a supplier for construction materials. The company outsources its production to contracted lumber suppliers in Asia and then uses capacitated ocean vessels to ship the semi-finished construction materials back to North America to serve regional markets. Before arriving at their destinations, semi-finished goods must be processed for customization and final configuration (e.g., trimming, sanding, sizing, labeling and packaging) at one of the processing centers (PCs) following customer specifications and requirements. After that, the finished goods are shipped to different regional markets (many demand points) via domestic trucking. Each regional market (a demand point) specifies its order quantity, quality grades, together with a delivery time (deadlines). Any delay in meeting the delivery time may result in lost sales, delay in construction projects, and thus customer dissatisfaction.

A simplified flowchart (i.e., with the second-tier suppliers and contracted operations at PCs omitted) of plywood/composite board production and shipping process is depicted below.

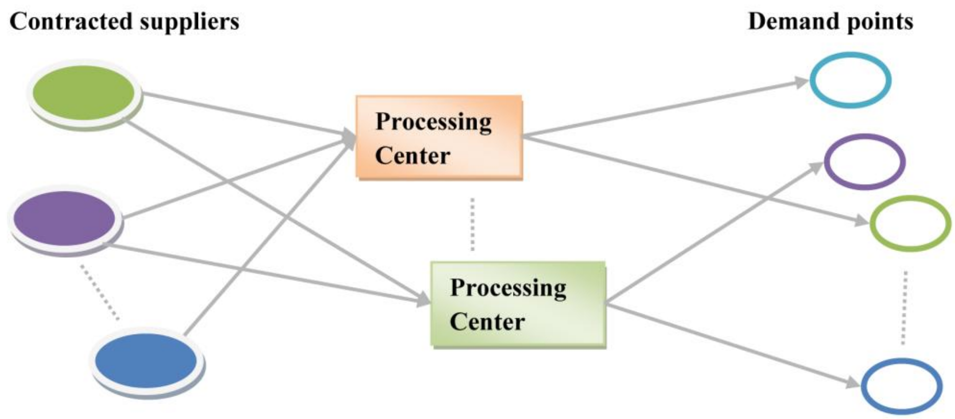

Figure 1.1 A two-echelon supply chain process for plywood

Another application on the multi-echelon supply chain networks was the logistics challenges faced by a company that designs/purchases and distributes various backpacks, luggage, handbags, and laptop socks, etc. The company outsources its production of semi-finished products/subassemblies, product accessories, and packaging materials to the suppliers in Asia, and then ships the goods back to North America to meet the seasonal demand (e.g., Mother's Day sales and back to school sales) of different regional markets. Before arriving at regional markets, the products must be finalized (and sometimes customized for individual vendors) and packaged at contracted processing centers (PCs). There are multiple groups of capacitated suppliers, and the suppliers in the same group are capable of producing the same supplies. Each business customer (i.e., distributor serving a regional market) specifies its order quantity, packaging specifications, and a delivery date (deadlines). Any failure in meeting the deadlines could result in lost sales and thus a penalty. In addition, there is a target order fill rate that the company prefers to retain for each sales region, which however may not always be achieved due to demand seasonality and capacity limits of both PCs and suppliers in practice. Whenever a target fill rate is missed, additional operation cost due to handling rush orders and a special shipping arrangement is imposed. Major operational issues that the company has been facing include, for any given set of customer orders from a given region, which customers should be notified if their orders cannot be delivered on time; which suppliers should be contracted, and how to assign suppliers and customer orders to the PCs, so that the total operation cost (shipping, manufacturing, and additional handling cost) is minimized. Since the inbound and outbound shipping and the packaging operations at the PCs are non-instantaneous, the resulting mathematical optimization problem is not a trivial one, especially when a large number of customer demand points are involved.

Such multi-echelon supply chain networks, involving original suppliers, multiple distribution centers with non-negligible processing times, and then many demand points, are usually more difficult to model and optimize than the single-stage shipping network, which can be typically handled as a variation of transportation problem (Miller, 1999). Effective multi-echelon operations scheduling in practices requires one to take into account both the shippers' capacities and processing centers' availability and to cope with the customer expectations in terms of time, costs, and reliability.

1.2 Integrated Supply and Distribution Problem

In this work, we are interested in the operations scheduling problem of the following integrated supply and distribution problem (ISDP). A semi-finished product needs to be shipped from a set of contracted manufactures to processing centers for their final configuration. After that, the finished products will then be shipped to many regional markets (demand points). The shipping network has three stages: suppliers (stage 1), processing centers or PCs (stage 2), and customer demand points (stage 3). Each Supplier has a limited production capacity and charges a shipping cost (via chartered vessels) proportional to the shipping quantity. Each PC has a limited capacity for processing the products. For any given PC, the processing starts only after all the shipment from assigned suppliers are received. It is assumed that each customer receives (customized) final products from no more than one PC, and the shipment arrival time at the customer site must be no later than its specified due date. If the shipment cannot arrive at a demand point at the specified due date (due to the delay in shipping and/or the limitation of network capacity), a penalty cost proportional to the order size placed by that customer is imposed. No partial delivery is acceptable. That is, if a customer receives the supply, then the order must be fully fulfilled. Both the shipping rate and the transit time from PCs to customer demand points are given. The objective is to determine the assignment of the demand points to the capacitated PCs and the operation schedules of transporters such that the total shipping and penalty cost is minimized.

A fully effective supply chain requires the integration of the front end of the supply chain (customer) with the back end of the supply chain (supplier). The global economic crisis of 2008 and 2009 provided significant disruptions and high demand volatility in supply chains for companies across many industries. As the global economy continues to recover, most of the companies now believe that there will be a significant upturn in demand from their customer base. Many companies, however, lack the capabilities critical for meeting growing demand or for managing an increasingly complex and global supply chain. To capture benefit from an eventual upturn, it depends largely on the implementation of effective supply chain operations strategies in increasing the total level of supply chain integration. The needs for effective decision tools that help to determine the supply chain operations decisions have been continuously increasing.

1.3 Summary of this Research

In this study, considering the delivery deadlines and customized products, we investigated the operations scheduling problem over the capacitated multi-echelon shipping networks. This problem is commonly encountered in the practices of outsourcing domestic production to low cost countries. Such operations scheduling problems, however, are also computationally difficult since they contain generalized assignment problems as sub-problems. To develop an algorithm to solve the general case of this problem, we proposed and proved mathematically that three special cases of this problem can be solved in polynomial time. The first case assumes that customer order sizes are identical and the shipping times from suppliers to PC are equal. It was proved that the original problem was equivalent to two sub problems: the transportation problem and its variant (i.e., when supply is not equal to demand), and the latter was solved in polynomial time due to its totally unimodular constraint matrix. The second one holds when each PC has its own exclusively designated supplier, and both suppliers and PCs have sufficient capacity. We showed that the original problem could be transformed into a Minimal Spanning Tree problem that is solvable in polynomial time. The third case relates to the assumptions that customer order sizes are divisible and all the operation costs (e.g., variable and fixed shipping cost) are customer-dependent. We showed the necessary conditions for this special case and proved mathematically that this case can be optimally solved in strongly polynomial time by the algorithm we developed. To solve the general case of this problem, we studied the operations scheduling problems with a single PC and multiple PCs, respectively. We proposed dynamic programming based search schemes that can successfully find the optimal subset of customer orders to be fulfilled under each given utilization level of the PC capacity and proved the resulting search algorithm convergence in pseudo-polynomial time. More importantly, we proposed a linear programming partial relaxation based algorithm to solve the problem with multiple PCs. This partial relaxation allowed us to avoid dealing with the generalized knapsack problem, a difficult NP-hard problem, in the solution process. With this algorithm, a given network shipping operations scheduling problem was solved through an iterative process.

This thesis is organized as follow. In chapter 2, a mathematical model for the operations scheduling problem of the capacitated multi-echelon shipping networks with delivery deadlines is proposed, and some literature on this problem and its methodology is reviewed. In chapter 3 we prove three polynomial time solvable cases: 1) Identical order quantities; 2) Designated suppliers; 3) Divisible order sizes and presented numerical examples to show the effectiveness of these cases. In chapter 4, we propose a dynamic programming based algorithm to solve the case where only PC is considered.

We prove that this algorithm can converge to the optimal solution within pseudo-polynomial time and use numerical example to show the solution process of this algorithm. Chapter 5 is dedicated to a linear programming based search algorithm that is designed to solve the general case of this problem. And then we conduct computational experiment to verify this proposed heuristic algorithm. Finally, we conclude the study and discuss future research directions in Chapter 6.3.

2 Multi-Echelon Operations Scheduling with Delivery Deadlines

In this chapter, we describe the operations scheduling problem for a capacitated multi-echelon shipping-network with delivery deadlines. We then formulate this problem as a mixed integer linear programming problem, based on the analysis of the multi-echelon supply chain operations, and discuss mathematical models and existing methodologies for solving this type of the operations problems in the literature.

2.1 Problem Statement

Over the network, semi-finished goods are shipped from a given set of origins (suppliers' sites) to many demand points (regional markets) through capacitated transshipment/ processing centers where the final configuration/customization of the product takes place. The shipping operations are performed by a fleet of capacitated transporters (vessels and tracks) which require a non-instantaneous time to move from one location to another. Each demand point has a specified a shipment quantity and a fixed deadline for delivery. Violating the deadline is not acceptable, and partial delivery is not considered. The problem is to find an operation schedule for the shipping network so that the network capacity constraints are satisfied while the sum of the shipping cost and the penalty cost due to missed delivery are minimized. This problem is commonly encountered in the practices of supply chains that outsource their production to low cost countries and then ship back the goods to meet the domestic demand. However, this problem is also a computationally difficult problem because of its inherent combinatorial nature.

2.2 Mathematical Model

To model this operations problem, we need to define the following decision variables used in our analyses:

x_{sn} = Amount shipped from supplier s to PC_n, $n \in \mathrm{N}$;

$$z_{sn} = \begin{cases} 1, \text{ if supplier } s \text{ ships to } \mathrm{PC}_n, s \in \mathrm{S}, n \in \mathrm{N} \\ 0, \text{ otherwise.} \end{cases};$$

$$y_{nj} = \begin{cases} 1, \text{ if } \mathrm{PC}_n \text{ supplies customer point } j, n \in \mathrm{N}, j \in \mathrm{J}, \\ 0, \text{ otherwise.} \end{cases}.$$

Our problem can now be formulated as the following mixed integer program (MIP).

$$\textbf{ISDP}: \quad \min \quad G = \sum_{s\in\mathbf{S}}\sum_{n\in\mathbf{N}} b_{sn}x_{sn} + \sum_{n\in\mathbf{N}}\sum_{j\in\mathbf{J}}\left(a_{nj}\lambda_j + \pi_{nj}\right)y_{nj} + \sum_{j\in\mathbf{J}} p_j\lambda_j\left(1-\sum_{n\in\mathbf{N}} y_{nj}\right)$$

s.t.

1. *Capacity constraints for suppliers:*

$$\sum_{n\in\mathrm{N}} x_{sn} \le F_s, s \in \mathrm{S} \tag{2.1}$$

$$x_{sn} \le z_{sn} F_s \, , s \in \mathrm{S} \, , n \in \mathrm{N} \tag{2.2}$$

2. *Flow balance constraints for PCs :*

$$\sum_{s \in S} x_{sn} = \sum_{j \in J} \lambda_j y_{nj} \, , n \in \mathrm{N} \tag{2.3}$$

3. *Each demand point accepts the supplies from at most one PC :*

$$\sum_{n \in N} y_{nj} \le 1, \; j \in \mathrm{J} \tag{2.4}$$

4. *Constraints on delivery deadlines, for all PC_n, $n \in \mathrm{N}$:*

$$\max\{t_{sn} z_{sn}\} + \sum_{s \in S} x_{sn} \tau_n + t_{nj} y_{nj} - M(1 - y_{nj}) \le T_j \, , \; j \in \mathrm{J} \tag{2.5}$$

5. *Constraints on the PC capacities:*

$$\sum_{j \in J} \lambda_j y_{nj} \le C_n \, , n \in \mathrm{N} \tag{2.6}$$

6. *Integrality and non-negativity:*

$$x_{sn} \ge 0 \, , \; z_{sn} \in \{0,1\}, y_{nj} \in \{0,1\}, \; s \in \mathrm{S} \, , n \in \mathrm{N} \, , j \in \mathrm{J} \tag{2.7}$$

The first term in the objective function of **ISDP** defines the shipping cost from suppliers to PCs **,** where $b_{sn} x_{sn}$ estimates the cost of shipping a quantity of x_{sn} from supplier s to $\mathrm{PC_n}$. The second term represents the variable and fixed shipping cost from PCs to demand points. The third term denotes the total penalty costs incurred by unsatisfied demands. Constraints (2.1) ensure that all the quantities shipped out of supplier s do not exceed supplier’s capacity while constraints (2.2) defines the relationship between the shipping quantities from supplier s to $\mathrm{PC_n}$ and the binary flow indicator variables z_{sn}, which equals to one if $x_{sn} > 0$. Constraints (2.3) are flow conservation constraints for each PC. Constraints (2.4) ensure that each demand point is supplied by at most one PC in accordance with the assumptions. Constraints (2.5) impose the requirement that the shipment arrives at demand point j must be no later than the specified due date. Constraints (2.6) ensure that PC capacities are not violated. Throughout the remaining discussion in this paper, we shall denote this problem as **ISDP**. **ISDP** can be shown to be NP-hard in strong sense, as it can be reduced to a dynamic generalized assignment problem (Kogan & Shtub, 1997) even if the deadlines of all demand points are relaxed and only a single supplier with infinity capacity is considered.

2.3 Literature Review

The problem of integrated capacitated network operations scheduling (**ICNOS**) has received an increasing attention during the past two decades because of the trend of outsourcing and globalization and the need to improve the operation efficiency by a highly collaborative and integrated production, inventory and distributions of supply chain networks. The relevant literature results can be classified into two categories: a). those involving integrated production-distribution scheduling (IPDS) and b). those with a focus on the integrated supply-distribution planning (ISDP) (i.e., without production or machine scheduling). A larger amount of research results in the first category can be found in the literature (Bhutta, K.S., 2003; Chang & Lee, 2004; Hall & Potts, 2005; Wang & Lee, 2005; Chen & Pundoor, Order assignment and scheduling in a supply chain, 2006; Lo, Wee, & Huang, 2007; Chiang, Russell, Xu, & Zepeda, 2009; Gebennini, Gamberini, & and Manzini, 2009; Rong, Akkerman, & and Grunow, 2011; Yan, Banerjee, & and Yang, 2011). Such models were proposed to find detailed order by order production and delivery schedules at the individual order level to optimize the tradeoffs between relevant revenues, costs and customer service levels. It is clear that the integrated production and distribution scheduling can significantly improve service and cost performance at the operational level.

On the contrary, the integrated supply-distribution planning focuses on coordinating the flows of supply and the demand of customers with respect to a given objective such as minimizing the sum of inventory, shipment, and shortage cost. Since our study is about a special case solution to the integrated supply-distribution planning problem over a capacitated multi-echelon network, we shall focus more on the literature in the second category. An early work in this area was done by Cohen & Lee (1988). Cohen and Lee (1988) studied a four-stage model with stochastic demands where the four stages included multiple vendors, multiple production plants, multiple DCs, and multiple customer zones. They formulated this problem as a decomposable mathematical program: material, production, inventory, and distribution sub-problems subject to a certain level customer service level. Then these four sub-problems were solved one on one to determine ordering policies which was able to minimize the total system-wide cost. Chandra & Fisher (1994) considered an integrated production and transportation planning problem with multi-products, a single production facility, multiple customers, and deterministic demand. This problem involved a setup cost for each production, inventory at both the plant and the customers, and transportation costs (variable and fixed costs).

They compared sequential and integrated approaches and obtained observations according to computational tests on randomly generated data for various parameters. Fumero & Vercellis (1999) investigated the similar problem to the one by Chandra and Fisher (1994). They considered a single-plant logistics system involved with a manufacturing unit and many retail outlets or peripheral depots. They then decomposed the problem into two sub-problems: a capacitated lot-sizing problem and a multi period vehicle routing problem and solved the proposed problem by Lagrangean relaxation. The comparison between decoupled approach and the integrated approach revealed that the optimal coordination of interrelated logistics decisions (e.g., capacity management, inventory allocation and vehicle routing) could be achieved through solving the dual master problem. Lei et al. (2006) considered an integrated production, inventory and distribution routing problem involving heterogeneous vessels with non-instantaneous traveling times, capacitated manufacturing facilities, and many customer demands (ocean terminals). They presented a two-phase solution approach where the first phase attempted to solve the direct shipment problem between manufacturing facilities and customers and showed that its optimal solution was always a feasible solution to the original problem. Then the second phase focused on a capacitated transportation problem with additional constraints used to supplement the potential inefficiency of direct shipments. Empirical studies demonstrated the potential improvement over classical decoupled approaches (i.e., separately solving the production and the transportation problems). Eksioglu et al. (2007) studied an integrated production and transportation planning problem in a two-stage supply chain. This supply chain consists of a number of facilities, each capable of producing the final products, and a number of retailers with deterministic demands. They formulated this problem as a multi-commodity network flow problem with fixed charge costs and production capacity constraints and proposed a Lagrangean-based decomposition approach to solve it. Computational tests were conducted on a large set of randomly generated problems to verify the quality of the lower and upper bounds of the solution that the proposed algorithm found. Bard et al (2009) developed a model that included a single production facility, a set of customers with time varying demand, and a fleet of vehicles. A reactive tabu search based solution approach which first solved an allocation model, as a mixed integer program, and used feasible solutions of the model as the starting points of tabu search. Computational testing on a set of 90 benchmark instances showed that 10-20\% improvements had been realized when compared to those from a greedy randomized adaptive search. Sawik (2009) considered a long-term integrated scheduling problem of material manufacturing, supply and assembly in a customer-driven supply chain. The supply chain consisted of three distinct stages: manufacturer/supplier of product-specific materials (parts), producer (assembled finished products), and a set of customers. The manufacturing stage consisted of identical production lines in parallel and the producer stage was a flexible assembly line. The problem was to coordinate manufacturing and supply of parts and assembly of products such that the total holding and shipping costs were minimized. Compared with a hierarchical approach, they used a monolithic approach (i.e., Schedule the manufacturing, supply and assembly simultaneously). Finally they presented numerical examples and reported some computational results. Zegordi & Beheshti Nia (2010) studied the integration of production and transportation scheduling in a two-stage supply chain environment with the assignment of orders to the suppliers. The first stage contains a set of suppliers distributed in various geographic zones, and the second stage consists of a fleet of vehicles with different speeds and transportation capacities. They presented a genetic algorithm to solve this problem and evaluated the performance of this proposed algorithm by comparing its outputs with optimum solutions for small-sized problems and to the random search approach for larger problems in addition to a similar problem from the literature. Amorima et al. (2012) studied an integrated production and distribution planning problem for highly perishable products that considered the intangible value of freshness. To find the integration advantages, they formulated this problem as multi-objective models for perishable goods with a fixed and a loose shelf-life. Their results showed that the economic benefits that an integrated approach results in strongly rely on the freshness level of products delivered. Bashiri et al. (2012) presented a new mathematical model for strategic and tactical planning in a multiple-echelon, multiple-commodity production–distribution network. In the proposed model, different time resolutions and expansion of the network are both considered for strategic and tactical decisions. To illustrate applications of the proposed model as well as its performance based on the solution times, some hypothetical numerical examples have been generated and solved by CPLEX. Results show that in small and medium scale of instances, high quality solutions can be obtained using this solver, but for larger instances, some heuristics has to be designed to reduce solution time. More importantly, to design the efficient heuristic algorithms for solving this multi-echelon operations scheduling problem, Lei & Wang (2012) reported three polynomial-time solvable cases of this problem with (a) identical order quantities; (b) designated suppliers; and (c) divisible customer order sizes. These analytical results revealed some interesting properties of the problem, and provided the theoretical foundation and initial solutions for the design of fast heuristic algorithms.

Our problem can be considered as a variation of the generalized transshipment problem with additional constraints (Guinet, 2001). While the classical transshipment problem has been well studied in the literature (Ahuja, Magnanti, & Orlin, 1993), its generalization to capacitated multi-echelon supply chain networks with additional constraints introduces new challenges for optimization. Yan et al. (2005) studied a transshipment problem with concave transportation cost and proposed a genetic algorithm along with a new encoding method to find a network flow solution by modifying infeasible flows into feasible ones. The proposed method was then evaluated against that of four local search algorithms developed upon the idea of threshold accepting criterion, great deluge heuristic, and tabu search. The results indicated that their proposed approach was more effective than local algorithms. Lei et al. (2009) studied a multi-period and multi-stage supply chain network optimization problem with both forward and reverse flows. An iterative partial-relaxation based search algorithm was proposed to schedule the amount of forward and reverse flows to minimize the total inventory and shipping cost. In each iteration, an optimization problem involving only the current time period, the next time period with partially relaxed integer variables, and a condensed period consisting of all future time periods with all integer variables relaxed, is solved. This partial relaxation allows one to solve a multi-period problem more quickly. Alpan et al. (2011) studied a transshipment scheduling problem encountered in a multiple-dock configuration. They presented three heuristics to solve the transshipment problem based on the solution space generated by a dynamic programming model for the same problem. Numerical experiments showed that, while these heuristics were well parameterized, they were able to find the near optimal solution much faster than dynamic programming. Kannegiesser & Günther (2011) presented a decision support tool for global value chain planning in the production of chemical commodities. They proposed a linear optimization model to reflect sales, distribution, production, and procurement activities. The objective of the model was to maximize profit by coordinating all activities within the supply chain. Then they explained the use of the model to support decision making from sales to procurement by volume and value. Bertazzi & Zappa (2012) studied the real case of Mesdan S.P.A., an Italian worldwide leader in the textile machinery sector. This company has two production units with two warehouses, one located in Italy (Brescia) and the other in China (Foshan), and owns different types of vehicles with different unit costs and extremely different lead times. They aimed to determine integrated policies between production and transportation management so as to minimize the total cost including variable production costs, variable and fixed transportation costs, and possible inventory costs. To this end, they formulated this problem as integer linear programming models, solved the real instance and performed a sensitivity analysis.

To our knowledge, most existing work that are related to our study does not take into account the assignment of suppliers to processing/distribution centers and the customer receiving deadlines. Our model has two fundamental differences from those in the literature of integrated supply-distribution planning. First, most related literature focus on allocating supplies from production facilities, inventory, and distribution centers to demand points, while we also need to consider assigning capacitated suppliers to serve the needs of the distribution centers (i.e., PCs in our case). Second, we have to face the customers with shipment receiving deadlines and the assignment to the distribution centers. More importantly, all available approaches to solving the problems in this area are developed upon either problem decomposition or heuristics. However, the solution to our problem requires a simultaneous optimization of assigning demand points to PCs, assigning PCs to suppliers, altogether with flow quantities subject to network capacities, processing time, shipping time, and delivery deadlines to minimize the total cost of shipping and shortage. While our problem is still a variation of those studied in the second category, the existing solution approaches cannot be directly extended to our case due to additional complexities introduced by its bin-packing type of constraints, see constraints (2.4), and customer receiving deadlines, see constraints (2.5) of model **ISDP**.

3 Polynomial-Time Solvable Cases

In this chapter, we report the three strongly polynomial-time solvable cases with a). Identical order quantities; b). Designated suppliers; and c). Divisible customer-order sizes. These results reveal the fundamental properties of the problem we study, and can be used to facilitate the design of fast heuristics for operations scheduling of capacitated shipping networks.

3.1 Case 1: Identical Order Quantities

In many applications, the customer order sizes are equal (e.g., each customer orders a full truckload of plywood panels or a tank of gasoline). For instance, Whirlpool has made a concerted effort to ship products in full truckloads rather than in multiple less-than-truckload shipments. Another example is a Houston-based food distributor, Sysco, for various meat and food products. At its new redistribution centers (i.e., processing centers), full truckloads are prepared and sent to its customers. When the customer order sizes are all equal, we have the case of identical order sizes, or $\lambda_j = \lambda,\ j \in \mathrm{J}$. Furthermore, we assume that the shipping times from suppliers to PCs can be approximated as a constant (note that we still allow the shipping time from PCs to customers to be arbitrary), then the following results holds.

Theorem 3.1 *Let* $K_n = \lfloor C_n / \lambda \rfloor$, $n \in \mathrm{N}$. *If* $t_{sn} = T$, $\lambda_j = \lambda$, a_{nj} , π_{nj} *are integral, and* $\sum_{\forall n} K_n = J$, *for all* $s \in \mathrm{S}$, $n \in \mathrm{N}$ *and* $j \in \mathrm{J}$ *, then the resulting special case of* **ISDP**, $\mathbf{P_{IOS}}$, *can be solved in strongly polynomial time.*

Proof. Our proof consists of two parts: a) $\mathbf{P_{IOS}}$ is decomposable and b) $\mathbf{P_{IOS}}$ can be solved in strongly polynomial time.

Part a) Under the given assumptions, **P** can be written as follows:

$$\mathbf{P_{IOS}}:\ \min\ G = \sum_{s \in \mathbf{S}} \sum_{n \in \mathbf{N}} b_{sn} x_{sn} + \sum_{n \in \mathbf{N}} \sum_{j \in \mathbf{J}} \left(a_{nj} \lambda_j + \pi_{nj} \right) y_{nj} + \sum_{j \in \mathbf{J}} p_j \lambda_j \left(1 - \sum_{n \in \mathbf{N}} y_{nj} \right)$$

$s.t.$

$$\sum_{n \in N} x_{sn} \le F_s, \qquad \forall s \in \mathrm{S} \tag{3.1}$$

$$\sum_{s \in S} x_{sn} = K_n \lambda, \qquad \forall n \in \mathrm{N} \tag{3.2}$$

$$T + K_n \lambda \tau_n + t_{nj} y_{nj} - M(1 - y_{nj}) \le T_j, \ \forall j \in \mathrm{J} \tag{3.3}$$

$$\sum_{n \in N} y_{nj} \le 1, \qquad \forall j \in \mathrm{J} \tag{3.4}$$

$$\sum_{j \in J} y_{nj} = K_n, \qquad \forall n \in \mathrm{N} \tag{3.5}$$

$$x_{sn} \ge 0, y_{nj} \in \{0,1\}, \qquad \forall s \in \mathrm{S}, n \in \mathrm{N}, j \in \mathrm{J} \tag{3.6}$$

Since $\mathbf{P_{IOS}}$ does not contain any constraint that has both variables x_{sn} and y_{nj}, for all $s \in \mathrm{S}, n \in \mathrm{N}, j \in \mathrm{J}$, it can be decomposed into the following two sub-problems, $\mathbf{P_1}$ and $\mathbf{P_2}$:

$$P_1: \quad \min \quad \sum_{s \in \mathbf{S}} \sum_{n \in \mathbf{N}} b_{sn} x_{sn} \qquad\qquad P_2: \quad \max \quad \sum_{n \in \mathbf{N}} \sum_{j \in \mathbf{J}} c_{nj} y_{nj}$$

$$s.t. \qquad\qquad\qquad\qquad\qquad s.t.$$

$$\sum_{n \in \mathrm{N}} x_{sn} \le F_s, s \in \mathrm{S} \qquad \text{and} \qquad \sum_{j \in \mathrm{J}} y_{nj} \le K_n, n \in \mathrm{N}$$

$$\sum_{s \in \mathrm{S}} x_{sn} = K_n \lambda, n \in \mathrm{N} \qquad\qquad \sum_{n \in \mathrm{N}} y_{nj} = 1, j \in \mathrm{J}$$

$$x_{sn} \ge 0, s \in \mathrm{S}, n \in \mathrm{N} \qquad\qquad y_{nj} \in \{0,1\}, n \in \mathrm{N}, j \in \mathrm{J}$$

where $c_{nj} = \begin{cases} a_{nj}\lambda_j + \pi_{nj} - p_j \lambda_j, \ if \ T + \lambda K_n \tau_n + t_{nj} \le T_j \\ 0, \ otherwise. \end{cases}$.

Part (b) To prove the complexity of $\mathbf{P_{IOS}}$, first note that $\mathbf{P_1}$ is a transportation problem and is thus solvable in $O(N(S+N)^2 \log(S+N))$ (Kleinschmidt & Schannath, 1995). We now prove that $\mathbf{P_2}$ can also be solved in strongly polynomial time due to the total unimodularity of tis constraint matrix. Thus, solving $\mathbf{P_2}$ is equivalent to solving its LP relaxation. To start, consider that the constraint matrix of $\mathbf{P}_2$ has $N \times J$ columns and $N+J$ rows and takes the form

$$\mathbf{A} = \begin{Bmatrix} \overline{1} & 0 & \cdots & 0 \\ 0 & \overline{1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \overline{1} \\ \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \end{Bmatrix},$$

where the $\overline{1}$ is a row vector with J 1's and $\mathbf{I}$ is an identity matrix. All but two entries in each column of $\mathbf{A}$ matrix are zero; the two nonzero entries are equal to 1. Let $R = \{i_1, \ldots, i_r\}$ be any subset of $\{1, 2, \ldots, N+J\}$. Divide set R into two disjoint subset R_1 and R_2 such that $R = R_1 \cup R_2$. Let $R_1 = \{i'_1, \ldots, i'_{r_1}\}$, $R_2 = \{i''_1, \ldots, i''_{r_2}\}$ and $r_1 + r_2 = r$. Since only two non-zero entries in each column of $\mathbf{A}$ are nonzero and the remaining entries are all equal to zero, we have that

$$\sum_{k \in R_1} a_{kj} \in \{0,1\} \text{ for all } j \in \mathbf{J}.$$

In the same way, we observe that

$$\sum_{k\in R_2} a_{kj} \in \{0,1\} \text{ for all } j \in \mathbf{J}.$$

Hence, $\sum_{k\in R_1} a_{kj} - \sum_{k\in R_2} a_{kj} \in \{-1,0,1\}$. According to Theorem 5.23 (Korte & Vyen, 2006), the constraint matrix $\mathbf{A}$ is totally unimodular. In addition, since all the nonzero right-hand sides are integers, the total unimodularity of the constraint matrix indicates that the optimal solution to the LP relaxation of $\mathbf{P}_2$ gives an optimal integral solution to $\mathbf{P}_2$ and the LP relaxation of $\mathbf{P}_2$ is known to be solvable in polynomial time (Korte & Vyen, 2006). Furthermore, note that the LP relaxation of $\mathbf{P}_2$ is obtained by relaxing binary variable y_{nj}, i.e., $0 \le y_{nj}$ for all n and j. Let $y_{nj} = \xi_{nj} / \lambda_j$ and then the LP problem can be transformed into a variant of transportation problem as shown below:

$$\mathbf{P}_2' \qquad \min \sum_{n\in\mathbf{N}} \sum_{j\in\mathbf{J}} \left(c_{nj} / \lambda \right) \xi_{nj}$$

$$s.t.$$

$$\sum_{j\in \mathrm{J}} \xi_{nj} = K_n \lambda, \quad \text{for } n \in \mathrm{N}, \tag{3.7}$$

$$\sum_{n\in \mathrm{N}} \xi_{nj} \le \lambda, \quad \text{for } j \in \mathrm{J}, \tag{3.8}$$

$$\xi_{nj} \ge 0, \quad \text{for all } n \in \mathrm{N} \text{ and } j \in \mathrm{J}. \tag{3.9}$$

It's well known that there is a strongly polynomial time solution to $\mathbf{P}_2'$ that can be obtained in $O(N(N+J)^2 \log(N+J))$ (Kleinschmidt & Schannath, 1995). Therefore, we can state that $\mathbf{P_{IOS}}$ can be solved in $O(N(m_1+N)^2\log(m_1+N))$, where $m_1=\max\{S,J\}$, which completes the proof. □

Under the above assumptions (i.e., $\sum_{n\in\mathbf{N}} K_n = J$), each processing center achieves its full capacity utilization so that that total inbound quantities from suppliers and outbound quantities to customers both amount to the processing capacity (C_n). This ensures that the assignment of suppliers to processing centers does not affect the assignment of processing centers to customers. Hence, this special case can be decomposed into two independent sub-problems: a transportation problem and a customer assignment problem. Due to the assumption of the full truckloads, the customer assignment problem has a total modular constraint matrix, which implies that this sub-problem can be solvable in polynomial time. Thus this special case is solved efficiently.

We now show an example of this special case with S=3, N=2 and J=3. The shipping rate, fixed cost and shipping time from PCs to customers (DPs) are given in Table 1. In addition, we assume that the suppliers' capacities are $F_1=13$, $F_2=9$, and $F_3=12$; the shipping time from each supplier to any PC is $t_{sn}=T=10$; the shipping rates from suppliers to PCs are $b_{11}=2$, $b_{12}=1$, $b_{21}=4$, $b_{22}=6$, $b_{31}=5$ and $b_{32}=3$; the capacity and processing times of PCs are $C_1=11$, $C_2=23$, $\tau_1=4$ and $\tau_2=6$; and the order sizes, penalty cost and receiving deadlines of demand points are $\lambda_1=\lambda_2=\lambda_3=10$, $p_1=100$, $p_2=200$, $p_3=250$ and $T_1=80$, $T_2=160$, $T_3=200$, respectively.

Table 3.1 Shipping rate, fixed shipping cost and shipping time from PCs to customers (DPs)

$a_{nj} / \pi_{nj} / t_{nj}$	**DP$_1$**			**DP$_2$**			**DP$_3$**		
PC$_1$	2	10	3	3	12	1	2	12	8
PC$_2$	4	14	6	5	15	5	7	9	2

Since the original problem is decomposable (Theorem), combining the optimal solutions to $\mathbf{P_1}$ and $\mathbf{P_2}$ leads to the optimal solution to $\mathbf{P_{IOS}}$**:** we first solve $\mathbf{P_1}$ to obtain $x_{11}^* = 1$, $x_{12}^* = 12$, $x_{21}^* = 9$, $x_{22}^* = x_{31}^* = 0$, $x_{32}^* = 8$, then solve $\mathbf{P}_2'$ for $y_{11}^* = y_{12}^* = y_{23}^* = 0$, $y_{13}^* = y_{21}^* = y_{22}^* = 1$, which together form the optimal solution to $\mathbf{P_{IOS}}$ with a minimum operations cost $G\left(x^*, y^*\right) = 225$.

According to a recent survey on parcel shipping and global trade management [*Recession Leads to Widespread Adoption of Lower Cost Shipping Options,2010*], about 96% percent of survey respondents made changes to their business plans in response to the ever increasing fuel prices, and targeted at using lower cost shipping options. Toward that end, collaborative distribution and full truck load (FTL) operations have started to gain an increasing amount of attention as effective cost reduction strategies. When the full load operations are implemented, the results in the section can certainly be used to guide the shipping network operations scheduling.

3.2 Case 2: Designated Suppliers

In this section, we consider the case where each processing center in our study has a designated supplier, or equivalently each supplier has its own exclusive subset of PCs to serve. Furthermore, we assume that the processing time at each PC is approximately a constant independent of the quantity processed and the capacity of both the designated supplier and the processing center are sufficiently large, then **ISDP** has an efficient solution. To formally state this result, we assume

(A_1) Each PC has its unique supplier, i.e., each supplier is designated to an exclusive subset of PCs; Let N_s be the subset of PCs served by supplier s and $\cup_{s\in\mathrm{S}}\mathrm{N}_s = \mathrm{N}$, $\mathrm{N}_s \cap \mathrm{N}_{s'} = \varnothing$, $\forall s, s' \in \mathrm{S}$.

(A_2) $\min\{F_s, C_n\} \geq \sum_{j\in\mathrm{J}} \lambda_j$, $\forall s \in \mathrm{S}, n \in \mathrm{N}$.

(A_3) The total processing time at PC_n is a constant, B_n, independent of flow quantity, $\forall n \in \mathrm{N}$.

Having designated suppliers for processing centers are common in the practices where processing centers are located over different geographical regions and supplied by local supplies, especially for bulky and heavy raw materials (e.g., farm products). This special case result has its potential applications such as supplier base rationalization and reduction. This strategy has been adapted widely in today's industry as an important business strategy to improve supply chain performance. With fewer suppliers, it is more likely that each customer is served via a sole-sourcing arrangement and thus designated supplier. It has been observed that such a strategy could achieve enhanced leverage, better communication and information sharing, reduced unit cost, better flexibility, and responsiveness, easier access to technology and innovations, improved delivery performance, decreased inventories and cost per unit, and better quality monitoring and management (Institute for Supply Management, 2010). For any shipping network with reduced supplier base and designated suppliers for each PC, the results derived can be used as a decision support tool for the operations planning.

Theorem 3.2 *If Assumptions A_1-A_3 hold, then the resulting instance of* **ISDP**, $\mathbf{P_{DS}}$ *is solvable in* $O((S+N+J)^2)$.

Proof: By Assumptions A_1-A_3, **P** is equivalent to the following problem:

$$\mathbf{P_{DS}}: \ \min \ G = \sum_{s\in\mathbf{S}}\sum_{n\in\mathbf{N}} b_{sn}x_{sn} + \sum_{n\in\mathbf{N}}\sum_{j\in\mathbf{J}}\left(a_{nj}\lambda_j + \pi_{nj}\right)y_{nj} + \sum_{j\in\mathbf{J}} p_j\lambda_j\left(1-\sum_{n\in\mathbf{N}} y_{nj}\right)$$

s.t.

$$x_{sn} \le z_{sn}F_s, \qquad \forall s\in \mathrm{S},\ \forall n\in \mathrm{N} \tag{3.10}$$

$$x_{sn} = \sum\nolimits_{j\in J}\lambda_j y_{nj}, \qquad \forall n\in \mathrm{N}_s \tag{3.11}$$

$$\max\{t_{sn}z_{sn}\} + B_n + t_{nj}y_{nj} - M(1-y_{nj}) \le T_j, \quad \forall j\in \mathrm{J}, \forall n\in \mathrm{N}_s \tag{3.12}$$

$$\sum\nolimits_{n\in N} y_{nj} \le 1, \qquad \forall j\in \mathrm{J} \tag{3.13}$$

$$x_{sn} = 0, \qquad \forall s\in \mathrm{S}, \forall n\in \mathrm{N} \tag{3.14}$$

$$x_{sn} \ge 0, z_{sn}, y_{nj} \in \{0,1\}, \quad \forall s\in \mathrm{S}, n\in \mathrm{N}, j\in \mathrm{J} \tag{3.15}$$

To show the correctness of this result, let us construct a network $G=(V,E)$: $V = \mathrm{S}\cup\mathrm{N}\cup\mathrm{J}$ and $E = \{(s,n),(n,j) \mid t_{sn}+B_n+t_{nj} \le T_j, \forall s\in\mathrm{S}, n\in\mathrm{N}_s,\ j\in\mathrm{J}\}$. Add a dummy node O and arcs linked with all suppliers. Note that, since we are considering a three-stage network and there are only arcs between different stages, the network $G=(V,E)$ can be constructed in $O(S \cdot N \cdot J)$. Solve the minimum spanning tree problem upon $G=(V,E)$, which runs in $O((S+N+J)^2)$ (Korte & Vyen, 2006). Then, the solution obtained has the following properties: (a) Each demand point is served by only one PC; (b) All the time constraints are satisfied; (c) All capacity constraints are satisfied due to Assumption A_2. □

Theorem 3.3 *If the assumptions A_2 and A_3 do not hold, then even if all the demand points have the same deadlines, i.e., $T_j = T$ for all $j \in J$, the problem remains NP-hard in strong sense.*

Proof: Even if all the demand point have the same deadlines, i.e., $T_j = T_{j'}$ for all $j, j' \in \mathrm{J}$, and even if we ignore suppliers, the assignment of demand points to PCs remains to be a dynamic generalized assignment problem. The dynamic generalized assignment problem is well-known to be NP-hard (Kogan & Shtub, 1997). □

To better understand this special case, let us consider the following instance of $\mathbf{P_{DS}}$ with S=2, N=4 and J=5. The shipping rate and travel time from suppliers to PCs, the shipping rate, travel time and fixed cost from PCs to customers, and customer orders and shown in Table 3.2-3.4, respectively. Other parameters are as follows: suppliers' capacity: $F_1 = 160$ and $F_2 = 140$; capacity and processing time of PCs: $C_1 = 70$, $C_2 = 72$, $C_3 = 82$,

$C_4 = 65$, $B_1 = 5$, $B_2 = 6$, $B_3 = 13$ and $B_4 = 11$. In addition, we have $N_1 = \{PC_1, PC_2\}$ and $N_2 = \{PC_3, PC_4\}$.

Since each supplier serves and exclusive subset of PCs, and the processing time at each PC is a constant, we can directly compute the total time from each supplier to each demand point as shown in Table 3.5.

Table 3.2 Shipping rate and travel time from suppliers to PCs

b_{sn}/t_{sn}	**PC_1**		**PC_2**		**PC_3**		**PC_4**	
S_1	2	10	3	13	0	0	0	0
S_2	0	0	0	0	2	12	4	9

Table 3.3 Shipping rate, fixed cost and shipping time form PCs to customers (DPs)

$a_{nj}/\pi_{nj}/t_{nj}$	**DP_1**			**DP_2**			**DP_3**			**DP_4**			**DP_5**		
PC_1	2	3	3	4	2	2	5	3	5	6	9	5	9	7	2
PC_2	4	4	5	5	7	3	4	5	4	6	6	4	8	2	5
PC_3	1	6	3	8	2	3	5	3	2	11	3	6	4	8	10
PC_4	2	2	2	2	5	12	4	1	3	4	5	3	1	4	7

Table 3.4 Demand quantities, deadlines and unit penalty cost of DPs

	DP_1	**DP_2**	**DP_3**	**DP_4**	**DP_5**
Demand (λ_j)	10	11	12	14	18
Deadline (T_j)	23	16	24	32	17
Penalty Cost (p_j)	100	200	250	210	200

Table 3.5 Total shipping time from suppliers to DPs through PCs

Shipping time	**DP_1**	**DP_2**	**DP_3**	**DP_4**	**DP_5**
S_1-PC_1	18	17	20	23	17
S_1-PC_2	24	22	23	21	24
S_2-PC_3	28	28	31	21	35
S_2-PC_4	22	32	18	23	27

Construct the shipping network, and apply the algorithm for solving the MST problem (Korte and Vyen, 2006), and we obtain the following optimal solution to $\mathbf{P_{DS}}$: $y^*_{11} = 1$, $y^*_{21} = y^*_{31} = y^*_{41} = 0$; $y^*_{12} = y^*_{22} = y^*_{32} = y^*_{42} = 0$; $y^*_{13} = 1$, $y^*_{23} = y^*_{33} = y^*_{43} = 0$; $y^*_{44} = 1$, $y^*_{14} = y^*_{24} = y^*_{34} = 0$; $y^*_{25} = 1$, $y^*_{15} = y^*_{35} = y^*_{45} = 0$ as well as $x^*_{11} = 22$, $x^*_{12} = 18$, $x^*_{13} = x^*_{14} = 0$, $x^*_{24} = 14$, $x^*_{21} = x^*_{22} = x^*_{23} = 0$, which results in a minimum cost of $G^* = 2647$.

3.3 Case 3: Divisible Order Sizes

We now consider a special case where we allow an arbitrary number of customers and heterogeneous PC capacities. However, customer order sizes must be divisible. Coffman et al. (1987) studied such a problem and focused on minimizing the number of identical bins with the same capacity. Detti (2009) discussed a multiple knapsack problem with divisible item sizes and presented a polynomial algorithm that run in $O(n^2 + nm)$, where n and m are the number of different item sizes and the number of knapsacks, respectively. We will extend the existing results on multiple knapsack problems with divisible item sizes to our three-stage shipping network problem with additional assignment (i.e., PCs to suppliers and customers to PCs) and customer receiving deadlines, which add new challenges into our solution process. In addition, we develop a necessary condition which ensures that our algorithm converges at the optimal solution to the problem.

In particular, we make the following assumptions:

E1) Order sizes are divisible, i.e., $d_i | d_{i+1}$ *for* $i = 1, 2, \cdots, m-1$.

E2) $b_{sn} = b$, $a_{nj} = a'_j$ *and* $\pi_{nj} = \pi'_j$ *for all* $s \in S$, $n \in N$ *and* $j \in \mathrm{J}$.

E3) $t_{sn} = T$, τ_n is a constant dependent of PCs, and $T_j \geq T + t_{nj} + \tau_n$ *for all* $s \in \mathrm{S}$, $n \in \mathrm{N}$ *and* $j \in \mathrm{J}$.

E4) Sufficient suppliers' capacity, i.e., $\sum_{s \in \mathrm{S}} C_s \geq \sum_{j \in \mathrm{J}} \lambda_j$.

E5) Let $v_j = p_j \lambda_j - a'_{nj} \lambda_j - \pi'_{nj} - b\lambda_j$ be the contribution of fulfilling order j and let M_i be the set of the remaining orders with size less than or equal to d_i that are not fulfilled by using residual capacity when the orders of size d_i are evaluated, and let W_i be any subset of d_{i+1}/d_i orders from M_i. Then $\sum_{k \in W_i} v_k \leq \min\{v_j \mid j \in \Phi_{i+1}\}$, $i = 1, 2, \cdots, m-1$.

These assumptions imply that the sizes of larger orders are integer multiples of those of smaller orders, shipping rate depends on only customer locations, the shipping deadlines are not binding constraints, the suppliers' capacities are sufficient for given customer demand points and that the total *savings* of any subset of smaller orders, with the sum of their sizes no more than d_i, is no more than the saving of any order in N_{i+1}. Not that Assumption E_1 is the strongest assumption in this case, which says that the size of a larger order, d_i, is always an *integer multiple* of a smaller order size, $d_{i'}$, $1 \leq i' < i \leq m$. Nevertheless, such an assumption does occur in real life supply chain practices. For example, lumber dying operations is done in batches within large kiln dryers according to the requirements specified by industry standards (Gaudreault, et al., 2010). Bundles of lumbers of different length can be dried in the same batch (e.g., 8- and

16 foot). A bundle must be assembled as a rectangular prism filling the kiln dryer almost entirely. Each sawmill defines its own set of loading patterns that can be used (see Fig.2). Another example is IKEA, a pioneer flat-pack furniture designer and distributor of approximately 10,000 products with 1,600 contracted suppliers and 27 distribution centers, has been a known leader in modular product design. Its product are typically in the size such as 16 $\times$16, 2 $\times$2, 4 $\times$4, etc., to meet various product design needs. IKEA can utilize the following algorithm proposed in the special case to efficiently group the products of smaller sizes into the ones with larger sizes, which facilitates operations planning for the production of final products and transportation. Then, let us start with the following results. Upon *E1)-E5)*, our original problem **ISDP** now reduces to the following mixed integer program:

$$\mathbf{P_{DOS}}: \quad \max \quad \sum_{s\in S}\sum_{n\in N} b x_{sn} + \sum_{n\in N}\sum_{j\in J}(a'_j\lambda_j + \pi'_j) y_{nj} + \sum_{j\in J} p_j \lambda_j (1 - \sum_{n\in N} y_{nj})$$

$$s.t.$$

$$\sum\nolimits_{n\in N} x_{sn} \le F_s, \ \forall s \in S, \tag{3.16}$$

$$\sum\nolimits_{s\in S} x_{sn} = \sum\nolimits_{j\in J} \lambda_j y_{nj}, \ \forall n \in N, \tag{3.17}$$

$$\sum\nolimits_{n\in N} y_{nj} \le 1, \quad \forall j \in J, \tag{3.18}$$

$$\sum\nolimits_{j\in J} \lambda_j y_{nj} \le C_n, \ \forall n \in N, \tag{3.19}$$

$$x_{sn} \ge 0, y_{nj} \in \{0,1\}, \ \forall s \in S, n \in N, j \in J. \tag{3.20}$$

Summing over index n for constraint (3.17), we have the following equivalence:

$$\sum_{n\in N}\sum_{s\in S} x_{sn} = \sum_{n\in N}\sum_{j\in J} \lambda_j y_{nj} \Leftrightarrow \sum_{s\in S}\sum_{n\in N} x_{sn} = \sum_{j\in J}\sum_{n\in N} \lambda_j y_{nj}$$

By the definition that $v_j = p_j\lambda_j - a'_{nj}\lambda_j - \pi'_{nj} - b\lambda_j$, $\mathbf{P_{DOS}}$ is equivalent to the following maximization problem:

$$\mathrm{P}'_{\mathrm{DOS}}: \qquad \max \quad \sum_{n\in \mathrm{N}}\sum_{j\in \mathrm{J}} v_j y_{nj} \tag{3.21}$$

$$\text{s.t.} \qquad (3.18),\ (3.19)$$

$$y_{nj} \in \{0,1\},\ \forall n \in \mathrm{N},\ j \in \mathrm{J} \tag{3.22}$$

which is a generalized knapsack problem and thus the focus of the remaining analysis in this section.

Let C_n^k be the remaining capacity of PC_n after the orders of size d_{k-1} have been evaluated and $r_n^k = (C_n^k \bmod d_{k+1})$ be the residual capacity for orders of size d_k, $k = 1,2,\cdots,m$, $n \in \mathrm{N}$. Let $h_k = \min\left\{n_k, \sum_{n=1}^{N} \left\lfloor r_n^k / d_k \right\rfloor\right\}$, and $\hat{\Phi}_k$ be the sequence of orders in non-increasing values of v_j during iteration k.

Theorem 3.4 *An optimal solution to* $\mathrm{P}'_{\mathrm{DOS}}$ *exists in which the first* h_k *orders in* $\hat{\Phi}_k$ *are assigned to the PCs using at most a capacity* r_n^k *from each* PC_n, $n \in \mathrm{N}$, $k = 1,2,\cdots,m$.

Proof: Let ϕ be the set consisting of the h_k orders in $\hat{\Phi}_k$, $k = 1,2,\cdots,m$. Note that, by the definition of h_k, an optimal solution to $\mathrm{P}'_{\mathrm{DOS}}$ contains at least h_k orders from set ϕ. Otherwise, we can always substitute an element in the solution with an unassigned order of the same size but a larger value of saving v_j, and improve the objective function value. On the other hand, if order $j \in \phi$ is not in an optimal solution to $\mathbf{P_{DOS}}$, but order j' $j' \notin \phi$, then substituting *j'* by *j* will lead to a solution with equal or better objective function value. Hence, all the orders in set ϕ must be in the optimal solution. Since $h_k = \min\left\{n_k, \sum_{n=1}^{N}\left\lfloor r_n^k / d_k \right\rfloor\right\}$ and residual capacity cannot be used for any orders of larger than d_k, all the orders in set ϕ can be always assigned using at most a residual capacity of each PC_n, $n \in \mathrm{N}$, $k = 1, 2, \cdots, m$. □

According to Theorem 4, we propose the following algorithm.

Algorithm A₁

Input: An instance of **P_{DOS}** $(S, N, J, F_s, C_n, \lambda_j, a'_j, b, \pi'_j, p_j)$;

Output: The optimal solution x^*_{sn} and y^*_{nj} that minimizes (3.21)

(1) Let the iteration index $k = 1$.

(2) While $k < m$, do

(2a) Rearrange the orders in Φ_k into a non-increasing sequence, $\hat{\Phi}_k$, in terms of v_j, and compute r_n^k and h_k. Select/assign the first h_k orders in $\hat{\Phi}_k$ to be fulfilled using at most a residual capacity r_n^k at each PC_n, $n \in \mathrm{N}$. For each assigned order j to PC_n, let y^*_{nj}=1.

(2b) Let $\theta_k = d_{k+1} / d_k$, . For the remaining orders in $\hat{\Phi}_k$, concatenate every consecutive θ_k orders to form a new composite order of size d_{k+1}. Add such composite orders, each of size d_{k+1}, to set Φ_{k+1}.

(2c) Set $k = k+1$, and $C_n^k = C_n^{k-1} - r_n^{k-1}$, $\forall n \in \mathrm{N}$.

(3) If ($k = m$)

Form the sequence $\hat{\Phi}_k$, assign the first h_k orders and discard the remaining orders.

(4) Given y^*_{nj} , apply (3.16) and (3.17) to solve for x^*_{sn} by Gaussian elimination method (Korte and Vyen, 2006), $\forall s \in \mathrm{S}, n \in \mathrm{N}, j \in \mathrm{J}$ **.**

Lemma 3.1 [*From Coffman et al. (1987)] If order sizes are divisible, then any set of orders that individually do not exceed d_i and that in total sum equal to at least must contain a subset that sums exactly to d_i , $k = 1, 2, \cdots, m$.*

Theorem 3.5 Algorithm A_1 always finds the optimal solution to $\mathbf{P_{DOS}}$.

Proof. Let $L = I_1, I_2 \ldots, I_n$ be the ordered list of orders that A1 found. The processing center PC_1 contains order I_1. Let PC be the corresponding processing center containing I_1. Choose optimal solutions that maximize the index k of the first order in L on which PC_1 and PC differ, with $k = \infty$ if the two processing centers are identical. Hence if $k = \infty$, we are done. Otherwise, assume that I_k belong to PC_1 and not PC. Now, by Lemma 3.1, either the sum of all the orders in PC with index smaller than k is less than I_k or some subset of those orders sums exactly to I_k . In either case, we can remove the corresponding orders from PC, replace them with I_k . But this creates a new optimal solution with larger value of k, contradicting our original choice of an optimal solution. Thus it must be $k = \infty$. Therefore, the solution of Algorithm A_1 is optimal. □

Theorem 3.6 *If an instance of* **ISDP** *,* $\mathbf{P_{DOS}}$ *, satisfies the assumptions E_1-E_5, then it can be solved in* $O(SN^3 + NJ + J^2 \log J)$.

*Proof***.** Our proof consists of two parts: a) Under the given conditions *E1)-E5),* problem $\mathbf{P_{DOS}}$ can be solved in polynomial time; and b) upon the optimal solution to $\mathbf{P_{DOS}}$, the optimal solution to **P** can be constructed in polynomial time.

Part a) Sorting in Step 1 requires $O(J \log J)$. Computing r_n and h_1 requires $O(n_1 + N)$. The number of demand point of size d_2 produced is $O(n_1 / f_2) = O(n_1 / 2)$ that follows from Detti (2009), where $f_2 = d_2 / d_1 \geq 2$. In fact, at least two demand points of size d_1 must be grouped to construct a demand point of size d_2. After assigning the demand points with size d_1, the smallest demand size is d_2, and $O(n_1 / f_2 + n_2)$ demand points of size d_2 exist. Hence, the iteration requires $O(n_1 / f_2 + n_2 + N) = O(n_1 / 2 + n_2 + N)$ operations and produces $O(n_1 / (f_2 f)_3 + n_2 / f_3) = O(n_1 / 4 + n_2 / 2)$ demand points of size d_3, where $f_3 = d_3 / d_2 \geq 2$. Repeating the above argument, we have that the last iteration starts with $O(\sum_{k=1}^{m-1} n_k / 2^{m-1-k})$ demand points of size d_{m-1} , and requires $O(N + \sum_{k=1}^{m-1} n_k / 2^{m-1-k})$ operations. Summarizing, we have

$$(m-1)N + \sum_{k=1}^{m-1} \sum_{i=1}^{m-k} \frac{n_k}{2^{i-1}} < (m-1)N + \sum_{k=1}^{m-1} \sum_{i=1}^{m-1} \frac{n_k}{2^{i-1}}$$

operations in total. Therefore, $\mathbf{P_{DOS}}$ can be solved in $O(NJ + J \log J)$ time.

Part b) Based on the y^*_{nj} obtained from $\mathbf{P_{DOS}}$, solving the linear systems of equations (3.16) and (3.17) can derive the value of x^*_{sn} by using Gaussian Elimination method (Korte & Vyen, 2006).

We now consider an instance of this special case with $S=4, N=3, J=10$, $m=3$, $T=5$, $b=5$, $F_1=7, F_2=10, F_3=11, F_4=13$ and $C_1=15, C_2=10, C_3=14$. The additional parameters regarding customer orders are presented in Table 3.6, which shows $\Phi_1=\{1,2,3,4,5\}$, $\Phi_2=\{6,7\}$ and $\Phi_3=\{8,9,10\}$.

Table 3.6 Order size, penalty cost, saving and variable and fixed cost

Order	**Size** λ_j	**Penalty Cost** p_j	**Shipping cost** a'_j	**Fixed shipping cost** π'_j	v_j
1	2	200	2	18	368
2	2	120	3	12	212
3	2	100	5	10	170
4	2	115	6	40	168
5	2	100	5	16	164
6	4	200	3	13	755
7	4	180	2	10	682
8	8	300	1	24	2328
9	8	210	4	15	1593
10	8	200	2	11	1533

In the Step 1 of Algorithm A$_1$, the orders in Φ_i are sequenced in non-increasing order of values, v_j, as shown in Table 3.6. During the iteration 1 of Step 2, we have $\hat{\Phi}_1 =< 1,2,3,4,5 >$, $r_1^1=3, r_2^1=2, r_3^1=2$, and thus $h_1=3$. Hence, order 1, 2 and 3 are assigned to three PCs and the remaining orders in $\hat{\Phi}_1$ are grouped to form the composite orders of size 4 with value 332, denoted as order 4-5. At the beginning of the second iteration, $C_1^1=13, C_2^1=8, C_3^1=12$ and $\Phi_2=\{6,7,4-5\}$, $\hat{\Phi}_2 =< 6,7,4-5 >$ with 755, 682 and 332, respectively. Since $r_1^2=5$, $r_2^2=0$, $r_3^2=4$ and $h_2=2$, both 6 and 7 in $\hat{\Phi}_2$ a re assigned to PC 1 and 3, respectively. The only remaining order 4-5 is then combined with a dummy order of value zero into a new order of size 8 with the value of 332. At the end of Step 2, we have $C_1^2 = 9, C_2^2 = 8, C_3^2 = 8$. In Step 3, we obtain that $h_3 = 3$ so that all the orders are assigned except for that composite order 4-5 with the value of 332. Table 3.7 summarizes the final solution, which shows that $y_{11}^* = y_{22}^* = y_{33}^* = y_{16}^* = y_{37}^* = y_{18}^* = y_{29}^* = y_{3,10}^* = 1$.

According to the given y_{nj}^*, we can use the Gaussian elimination method to simultaneously solve (3.16) and (3.17) and obtain x_{sn}^* : $x_{11}^* = 2$, $x_{21}^* = x_{31}^* = x_{41}^* = 4$, $x_{12}^* = x_{22}^* = 2$, $x_{32}^* = x_{42}^* = 3$, $x_{13}^* = x_{23}^* = x_{33}^* = 3$ and $x_{43}^* = 5$. Hence, we have that the total cost $G^* = 829$.

4 Dynamic Programming-based Algorithm with A Single PC

In this chapter, we solve a special case of **ISDP** when the number of processing centers (PC) equals to one, and develop a dynamic programming based search algorithm which identifies the optimal subset of customer orders to be fulfilled under each given utilization level of the PC capacity. We then construct a cost function of a recursive form, and prove that the resulting search algorithm always converges to the optimal solution within pseudo-polynomial time. Note that this single PC in our study may be interpreted as a single production facility or a single distribution center, which is designated to serve the customers in a given region. Note that the problems with a single PC or DC serving a given region are very common in practices. For example, Americares, a nonprofit disaster relief and humanitarian aid organization, has only three distribution centers/warehouses for its global operations; each serves the demand from a particular country (i.e., US, India, and Haiti). Cardinal Health Inc., a pharmaceutical product wholesaler, has only one distribution center serving all the demand points in Alaska. This result can also serve as a subroutine embedded in a more general heuristic that solves the supply chain operation s scheduling problem with multiple PCs.

4.1 Problem Description

We consider a three-stage supply chain process (see Figure 1) consisting of heterogeneous groups of suppliers, a single processing center (PC), and a network of customer demand points. Assumptions about suppliers, the PC, and customers, upon which we shall construct the mathematical model, are as follows.

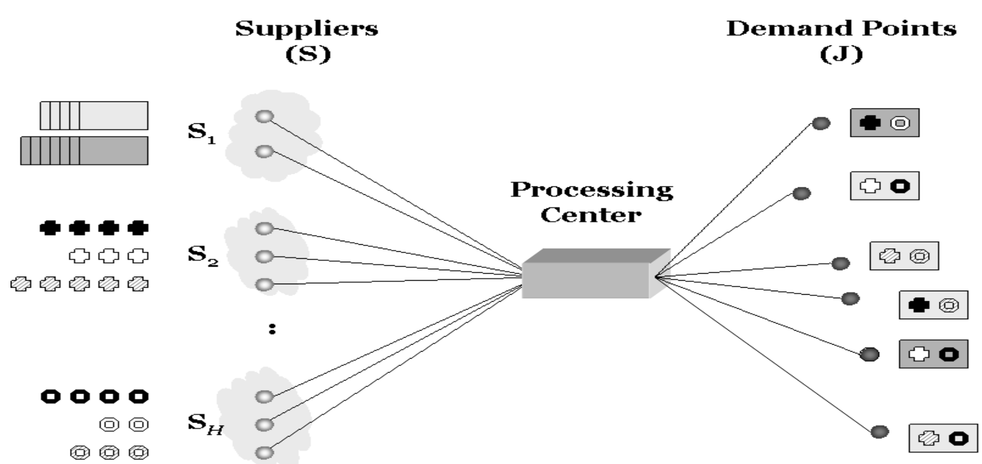

Figure 4.1 A three-stage supply chain process

Customers: Each customer *j*, $j \in \mathrm{J}$, has an order which specifies the delivery deadline (T_j) and order quantity (λ_j). Since the network is capacitated, not all the customers may receive their shipment by the deadline and thus some customer orders will be missed (and then handled differently). However, if customer *j* is selected to receive the shipment, then a shipment of λ_j units of customized product must arrive at the customer site no later than T_j. Otherwise, a customer-dependent penalty due to missed delivery, $p_j \cdot \lambda_j$, is imposed.

Suppliers: Let *H* denote the total types of distinct functional components used for processing at the PC, and let S_h denote the set of candidate suppliers specialized in making component *h*, $h \in \mathrm{H} = \{1, 2, \cdots, H\}$. Let $\mathrm{S} = \bigcup_{h \in \mathrm{H}} \mathrm{S}_h$ stand for the set of suppliers in the network. Each candidate supplier s, $s \in \mathrm{S}$, has a limited production capacity, F_s, charges a fixed shipping rate to the PC, b_s (unit), and requires a lead/shipping time to deliver to the PC, $t_s^{\sup}$, $s \in \mathrm{S}$.

PC: The PC has a limited capacity, *C*, for finalizing the products and requires a non-negligible processing time, τ, for each unit of product to be finalized. Let t_j^{DP} be the required shipping time from the PC to customer *j*, let a_j be the shipping rate, and π_j be the fixed cost if there is any shipment, from the PC to customer *j*. It is assumed that the PC does not have any initial inbound and outbound inventory, and serves only as a processing center. It is also assumed that the processing at the PC will not start until all the required components (i.e., raw materials) have arrived.

In addition, there is a non-linear penalty that applies whenever the total amount of order quantity delivered to the customer network is below a given target fulfillment rate β, $0 \le \beta \le 100\%$. We assume this penalty cost as follows

$$\gamma \cdot [\max\{0, \beta \cdot \sum_{j \in \mathrm{J}} \lambda_j - \sum_{j \in \mathrm{J}} \lambda_j \cdot y_j\}]^{\omega}, \qquad (4.1)$$

where $\gamma > 0$, $\omega > 1$, are penalty parameters and can take any positive values greater than 1, and $y_j \in \{0,1\}$, $j \in \mathrm{J}$, are binary variables where $y_j = 1$ if the order of customer j is fulfilled on time; and $y_j = 0$ otherwise, so that $\sum_{j \in \mathrm{J}} \lambda_j \cdot y_j$ is the total quantity fulfilled for the network. Quantity $\beta \cdot \sum_{j \in \mathrm{J}} \lambda_j$ stands for the minimum quantity to fulfill to meet the target fill rate β. The nonlinear penalty cost here defines the additional expenses incurred for handling rush orders that missed the delivery deadlines.

Our problem is then to select a subset of customer orders to be fulfilled on-time, and to select candidate suppliers from each group to support the PC production for the selected customers such that the sum of the shipping, processing, and penalty cost is minimized while all the capacity and delivery deadline constraints for selected customers are satisfied.

Let

x_s = The total shipping quantity from supplier s to the PC, $s \in \mathrm{S}$;

z_s = Binary variables, and $z_s = 1$ if supplier s is selected; and 0 otherwise, $s \in \mathrm{S}$;

y_j = Binary variables, and y_j =1 if the order of customer j is fulfilled on time; and 0 otherwise, $j \in \mathrm{J}$.

The problem that we are interested in this study can now be formally defined as follows:

Single_ISDP:

$$\min \ G = \sum_{h \in \mathrm{H}} \sum_{s \in \mathrm{S}_h} b_s x_s + \sum_{j \in \mathrm{J}} (a_j \lambda_j + \pi_j) y_j + \sum_{j \in \mathrm{J}} p_j \lambda_j (1 - y_j) + \gamma \cdot [\max\{0, \beta \cdot \sum_{j \in \mathrm{J}} \lambda_j - \sum_{j \in \mathrm{J}} \lambda_j \cdot y_j\}]^{\omega}$$

s.t.

1. *Capacity constraints on suppliers:*

$$x_s \le z_s F_s, \quad \forall s \in \mathrm{S} \tag{4.2}$$

2. *Flow balance constraints on the PC:*

$$\sum_{s \in \mathrm{S}_h} x_s = \sum_{j \in \mathrm{J}} \lambda_j y_j, \quad \forall h \in \mathrm{H} \tag{4.3}$$

3. *Constraints on delivery deadlines:*

$$\max_{s \in \mathrm{S}} \left\{ t_s^{\sup} z_s \right\} + \tau \sum_{l \in \mathrm{J}} \left(\lambda_l y_l \right) + t_j^{DP} y_j - M(1 - y_j) \le T_j, \ \forall j \in \mathrm{J} \tag{4.4}$$

4. *Constraint on the PC capacity:*

$$\sum_{j \in \mathrm{J}} \lambda_j y_j \le C \tag{4.5}$$

5. *Integrality and non-negativity:*

$$x_s \geq 0,\ z_s \in \{0,1\}, \forall s \in \mathrm{S},\ \text{and}\ y_j \in \{0,1\}, \forall j \in \mathrm{J} \tag{4.6}$$

The first, and the second, term in the objective function define the shipping cost from suppliers to the PC, and the fixed and variable shipping costs from the PC to customer demand points, respectively. The third term denotes the total penalty cost incurred by missed customer orders and the fourth term is a nonlinear penalty cost due to the shortage (if any) to meet the target fill rate (β) for the sales network. Constraints (4.2) ensure that all the quantities shipped out of a supplier do not exceed its capacity and that the relationship between the shipping quantity from a supplier to the PC and its flow indicator variable is correctly established. Constraints (4.3) guarantee that incoming flow quantity of component *h*, $h \in \mathrm{H}$, at the PC matches the required amount to meet the selected orders to be fulfilled. While we assume in this model a one-to-one conversion ratio that exactly one unit of each component is used in each unit of final product, this assumption can be easily extended to where multiple units of certain component are needed to make one unit of a finalized/customized product. Constraints (4.4) impose the requirement that the shipment arrives at a demand point must be no later than its specified delivery deadline. Finally, constraint (4.5) ensures the usage of the PC capacity to be within its maximum limit.

Single_ISDP remains to be a NP-hard problem even with a single component (i.e. *H* = 1), instantaneous transportation (i.e., $b_s = t_s^{\sup} = t_j^{DP} = 0$), no penalty for the shortage ($\gamma = 0$), and completely relaxed T_j and F_s. In this case, **Single_ISDP** can be reduced to the classical knapsack problem (Brotcorne, Hanafi, & Mansi, 2009), which is known to be NP hard, with a knapsack capacity *C*, item sizes λ_j and item values $p_j\lambda_j - a_j\lambda_j - \pi_j$, $\forall j \in \mathrm{J}$. While the classical knapsack problem is solvable in pseudo-polynomial time by dynamic programming (Kellerer & Pferschy, 2004), the exiting literature results cannot be directly applied to **Single_ISDP** due to its multi-echelon network structure, travelling time requirement, non-instantaneous processing operations at the PC, and customer receiving deadlines. Furthermore, the departure time of finished orders from the PC to customers is a variable whose value depends on i). which suppliers are selected, and ii). the total quantity to be processed, and thus the amount of processing time needed, at the PC. The algorithm that we shall propose for solving **Single_ISDP** must be able to simultaneously optimize the selection of suppliers, the quantity to be shipped from each selected supplier to the PC, and the selection of customers to receive the shipments on time. Solving **Single_ISDP** requires us to make the optimal tradeoff among production quantity (and thus the required processing time and potential delays in shipping), supplier selection (and thus the shipping cost, the required shipping time, and the PC starting time of processing), and the customer selection (and thus the penalty cost and the service level for the network), while subject to all the constraints. The algorithm that we shall propose in this study for solving **Single_ISDP** must be able to simultaneously optimize the integrated supply, manufacturing, and distribution operations. We are interested in solving this problem in pseudo-polynomial time, and shall propose such a search algorithm in the following section.

4.2 A Dynamic Programming-based Algorithm for Solving Single_ISDP

In this section, we focus on the solution approach for solving **Single_ISDP**. To start, let's arrange suppliers in an *increasing* order of their shipping times to the PC, i.e., $t_1^{\sup} \le t_2^{\sup} \le \cdots \le t_s^{\sup}$, and then customers in a *decreasing* order of their latest departure times (for the on-time shipment) from the PC, i.e., $T_1 - t_1^{DP} \ge T_2 - t_2^{DP} \ge .... \ge T_J - t_J^{DP}$, $\forall j \in \mathrm{J}$. Let $\mathrm{S}(\alpha)$ be the set of first α suppliers, i.e., $\mathrm{S}(\alpha) = \{1, 2, ..., \alpha\}$, and $\mathrm{S}_h(\alpha)$ be defined by $\mathrm{S}_h \bigcap \mathrm{S}(\alpha)$.

Let us define a sub-problem of **P**, $P^{\alpha\phi}$, $1 \le \alpha \le S$, $1 \le \phi \le J$, which involves the first α suppliers, the first ϕ customers, and a revised PC capacity $C(\alpha,\phi)$ defined as

$$C(\alpha,\phi) = \min\left\{C, \min_{h\in\mathrm{H}}\left\{\sum_{s\in\mathrm{S}_h(\alpha)} F_s\right\}, \sum_{j=1}^{\phi}\lambda_j, \left\lfloor \frac{T_\phi - t_\phi^{DP} - t_\alpha^{\sup}}{\tau} \right\rfloor \right\} \tag{4.7}$$

where $\left\lfloor (T_\phi - t_\phi^{DP} - t_\alpha^{\sup}) / \tau \right\rfloor$ gives an upper bound on the maximum quantity that the PC can produce between its earliest possible starting time and the latest feasible completion time under the given α and ϕ. We exclude all such sub-problems $P^{\alpha\phi}$ if $T_\phi - t_\phi^{DP} < t_\alpha^{\sup}$ and/or $C(\alpha,\phi) = 0$. Note that, in case the first α suppliers are not able to supply all the distinct H components, then, by definition of $C(\alpha,\phi)$, the revised capacity is zero and thus a trivial solution that selects no customers becomes the unique solution. Also note that for any given sub-problem $P^{\alpha\phi}$, the customers in set $\{\phi+1, \phi+2, \ldots, J\}$, are not served and thus penalized. An important advantage of introducing such a revised processing capacity of the PC is that the deadline constraints are always satisfied for the given subset of customers. We shall therefore remove constraints (4.4) from the analysis of $P^{\alpha\phi}$.

For a given pair of α and ϕ, $1 \le \alpha \le S$, $1 \le \phi \le J$, $P^{\alpha\phi}$ can be rewritten in the following:

$$P^{\alpha\phi}: \quad \min \ G^{\alpha\phi} = \sum_{h \in \mathrm{H}} \sum_{s \in \mathrm{S}_h(\alpha)} b_s x_s + \sum_{j=1}^{\phi} \left(a_j \lambda_j + \pi_j \right) y_j + \sum_{j=1}^{\phi} p_j \lambda_j (1 - y_j) + \sum_{j=\phi+1}^{J} p_j \lambda_j + \gamma \cdot [\max\{0, \beta \cdot \sum_{j=1}^{J} \lambda_j - \sum_{j=1}^{\phi} \lambda_j \cdot y_j\}]^{\omega}$$

$$s.\,t. \qquad x_s \le F_s, \qquad \forall s \in \mathrm{S}(\alpha) \tag{4.8}$$

$$\sum_{s \in \mathrm{S}_h(\alpha)} x_s = \sum_{j=1}^{\phi} \lambda_j y_j, \qquad \forall h \in \mathrm{H} \tag{4.9}$$

$$\sum_{j=1}^{\phi} \lambda_j y_j \le C(\alpha, \phi), \tag{4.10}$$

$$x_s \ge 0, \ y_j \in \{0,1\}, \qquad \forall s \in \mathrm{S}(\alpha), \qquad \forall j \in \{1, 2, \cdots, \phi\} \tag{4.11}$$

Now let us further consider a sub-problem of $P^{\alpha\phi}$, denoted by $P_j^{\alpha\phi}(d)$, in which only the first j customers can be served, $j = 1, 2, \ldots, \phi$, for order fulfillment by utilizing exactly d units, $0 \le d \le C(\alpha, \phi)$, of the PC capacity. Note that since the time constraints are no longer needed for $P^{\alpha\phi}$, the optimality of the supplier selection, for any given $P_j^{\alpha\phi}(d)$, depends only upon the values of α and d, but not ϕ. Therefore, we can separately solve the supplier selection problem under any given α and d. Furthermore, for any given PC capacity level d, $0 \le d \le \min\left\{C, \min_{h\in\mathrm{H}}\left\{\sum_{s\in\mathrm{S}_h(\alpha)} F_s\right\}\right\}$, to be utilized and the subset of suppliers $\mathrm{S}(\alpha)$, let $P^\alpha(d)$ denote the following transportation problem from the suppliers in $\mathrm{S}(\alpha)$ to the PC and $G^\alpha(d)$ be its optimal objective function value:

$$P^\alpha(d): \qquad G^\alpha(d) = \min \sum_{h\in\mathrm{H}} \sum_{s\in\mathrm{S}_h(\alpha)} b_s x_s$$

$$s.t. \qquad \sum_{s\in\mathrm{S}_h(\alpha)} x_s = d,\ \forall h \in \mathrm{H}, \quad x_s \le F_s, \quad x_s \ge 0, \ \forall s \in \mathrm{S}(\alpha)$$

where $G^\alpha(d)$ defines the minimum cost of shipping d units of supplies from suppliers in each subset $\mathrm{S}_h(\alpha)$, $h = 1, 2, ..., H$, to the PC. Moreover, $P^\alpha(d)$ can be decomposed into the following H independent sub-problems $P_h^\alpha(d)$, each aims to minimize the inbound shipping cost for component h, $G_h^\alpha(d)$, $h = 1, 2, \ldots, H$:

$$P_h^\alpha(d): \ G_h^\alpha(d) = \min \sum_{s\in\mathrm{S}_h(\alpha)} b_s x_s \quad s.t. \quad \sum_{s\in\mathrm{S}_h(\alpha)} x_s = d, \quad x_s \le F_s, \quad x_s \ge 0, \ \forall s \in \mathrm{S}_h(\alpha)$$

such that $G^\alpha(d) = \sum_{h\in\mathrm{H}} G_h^\alpha(d)$. In order to solve $P_h^\alpha(d)$, we sort suppliers in set $\mathrm{S}_h(\alpha)$ in a non-decreasing order of their unit shipping costs and let $s_1, s_2, ..., s_{|\mathrm{S}_h(\alpha)|}$ be such an ordering of elements in $\mathrm{S}_h(\alpha)$, i.e., $b_{s_1} \le b_{s_2} \le \cdots \le b_{s_{|\mathrm{S}_h(\alpha)|}}$. If $\sum_{\mu=1}^{\nu-1} F_{s_\mu} < d \le \sum_{\mu=1}^{\nu} F_{s_\mu}$, then

$G_h^\alpha(d) = G_h^\alpha(d-1) + b_{s_\nu}$. Thus, $G_h^\alpha(d)$ can be computed by increasing the value of d by one unit at a time, starting with d = 0, with the initial value of $G_h^\alpha(0) = 0$, and updating the value of ν accordingly. Therefore, with a given α, it takes $O(C)$ time to obtain all values of $G_h^\alpha(d)$ for all $0 \le d \le \min\left\{C, \sum_{s \in S_h(\alpha)} F_s\right\}$. Since we only need to sort the S suppliers in a non-decreasing order of their unit shipping costs once at the beginning and then use the first α suppliers in the sorted supplier list each time, it takes $O(S \log S + CS)$ time to obtain all the values of $G^\alpha(d) = \sum_{h \in H} G_h^\alpha(d)$ for all $1 \le \alpha \le S$ and $0 \le d \le \min\left\{C, \min_{h \in H}\left\{\sum_{s \in S_h(\alpha)} F_s\right\}\right\}$. Note that $G^\alpha(d)$ is the supplier-dependent fixed cost for the respective $P_j^{\alpha\phi}(d)$. Hence, $G^\alpha(d)$ and x_s, $\forall s \in S(\alpha)$, are handled as constants in the process of solving $P_j^{\alpha\phi}(d)$, and all the constraints associated with x_s can be removed from $P_j^{\alpha\phi}(d)$. For any given α, ϕ, j, $G^\alpha(d)$, and a capacity level to be utilized, d, the sub-problem $P_j^{\alpha\phi}(d)$ can be formally defined as follows:

$$\min\ z_j^{\alpha\phi}(d) = G^\alpha(d) + \sum_{k=1}^{j}(a_k\lambda_k + \pi_k)y_k + \sum_{k=1}^{j} p_k\lambda_k(1-y_k) + \sum_{k=j+1}^{J} p_k\lambda_k + \gamma \cdot [\max\{0, \beta \cdot \sum_{k=1}^{J}\lambda_k - \sum_{k=1}^{j}\lambda_k \cdot y_k\}]^\omega$$

$s.\ t.$

$$\sum_{k=1}^{j} \lambda_k y_k \le d, \tag{4.12}$$

$$y_k \in \{0,1\}, \qquad \forall k \in \{1,2,\cdots,j\}. \tag{4.13}$$

which is a standard knapsack problem and is known to be solvable by dynamic programming (Garey & Johnson, 1979). To construct the network for dynamic programming, let each stage be an individual customer *j*, $1 \le j \le \phi$, and each state at a given stage be a specific utilized PC capacity level, *d*, $d = 0,1,2,\cdots,C(\alpha,\phi)$. Then the recursive equation for the cost function of $P_j^{\alpha\phi}(d)$, $z_j^{\alpha\phi}(d)$, can be defined as

$$z_j^{\alpha\phi}(d) = \begin{cases} z_{j-1}^{\alpha\phi}(d) & \text{if } d < \lambda_j \\ \min \begin{cases} z_{j-1}^{\alpha\phi}(d), \\ \\ z_{j-1}^{\alpha\phi}(d-\lambda_j) + (a_j\lambda_j + \pi_j - p_j\lambda_j) + (G^{\alpha}(d) - G^{\alpha}(d-\lambda_j)) \\ +\gamma \cdot \{[\max\{0,\ \beta \cdot \sum_{k=1}^{J} \lambda_k - d\}]^{\omega} - [\max\{0,\ \beta \cdot \sum_{k=1}^{J} \lambda_k - d + \lambda_j\}]^{\omega}\} \end{cases} & \text{, otherwise} \end{cases} \tag{4.14}$$

with boundary conditions of $z_0^{\alpha\phi}(d) := \sum_{j=1}^{J} p_j\lambda_j + \gamma \cdot (\beta \cdot \sum_{j=1}^{J} \lambda_j)^{\omega}$ for $\alpha \in \mathrm{S}$, $\phi \in \mathrm{J}$, $d = 0,1,....,C(\alpha,\phi)$, and $z_j^{\alpha\phi}(0) := \sum_{j=1}^{J} p_j\lambda_j + \gamma \cdot (\beta \cdot \sum_{j=1}^{J} \lambda_j)^{\omega}$ for $\alpha \in \mathrm{S}$, $\phi \in \mathrm{J}$, $j \in \{1,...,\phi\}$.

Let G^* be the global minimum of the objective function. Our proposed search algorithm, called **SPO**, that utilizes the revised PC capacity, $C(\alpha,\phi)$, together with a dynamic programming based subroutine for solving $P_j^{\alpha\phi}(d)$, for each given combination of α, ϕ, *j*, and *d*, can be outlined as follows.

Algorithm SPO

Input: S, H, J and $F_s, t_s^{\sup}, b_s$, C, τ, λ_j, T_j, t_j^{DP}, π_j, a_j, p_j, γ, ω for $\forall s \in \mathrm{S}$, $\forall j \in \mathrm{J}$;

Output: The optimal solution to **P**, x_s^*, z_s^* and $y_j^*, \forall s \in \mathrm{S}$, $\forall j \in \mathrm{J}$;

Step 1. Order suppliers and customers.

Sort suppliers in an increasing order of their shipping times to the PC, $t_s^{\sup}$, $\forall s \in \mathrm{S}$; sequence customers in a decreasing order of their latest departure times from the PC for the on-time delivery, $T_j - t_j^{DP}$, $\forall j \in \mathrm{J}$.

Step 2. Solve the transportation problem $P^{\alpha}(d)$ for x_s^*.

For $\alpha := H$ to S

For $d := 0$ to $\min\left\{C, \min_{h \in \mathrm{H}}\left\{\sum_{s \in \mathrm{S}_h(\alpha)} F_s\right\}\right\}$,

Solve $P^{\alpha}(d)$ for $\{x_s^*\}$.

Step 3. Seek for the optimal network operations schedules.

a) Let $z_0^{\alpha\phi}(d) := \sum_{j=1}^{J} p_j \lambda_j + \gamma \cdot (\beta \cdot \sum_{j=1}^{J} \lambda_j)^{\omega}$ for $\alpha \in \mathrm{S}$, $\phi \in \mathrm{J}$, $d = 0,1,\ldots,C(\alpha,\phi)$ and $z_j^{\alpha\phi}(0) := \sum_{j=1}^{J} p_j \lambda_j + \gamma \cdot (\beta \cdot \sum_{j=1}^{J} \lambda_j)^{\omega}$ for $\alpha \in \mathrm{S}$, $\phi \in \mathrm{J}$, $j \in \{1,\ldots,\phi\}$. Set $G^* := \infty$.

b)

For $\alpha := H$ **To** S

For $\phi := 1$ **To** J

For $j := 1$ **To** ϕ

For $d := 1$ **To** $C(\alpha,\phi)$

If $d < \lambda_j$, **Then** $z_j^{\alpha\phi}(d) := z_{j-1}^{\alpha\phi}(d)$

Else

$$z_j^{\alpha\phi}(d) := \min\begin{cases} z_{j-1}^{\alpha\phi}(d), \\ z_{j-1}^{\alpha\phi}(d-\lambda_j) + (a_j\lambda_j + \pi_j - p_j\lambda_j) + (G^{\alpha}(d) - G^{\alpha}(d-\lambda_j)) \\ +\gamma \cdot \{[\max\{0,\ \beta \cdot \sum_{k=1}^{J} \lambda_k - d\}]^{\omega} - [\max\{0,\ \beta \cdot \sum_{k=1}^{J} \lambda_k - d + \lambda_j\}]^{\omega}\}. \end{cases}$$

$z_{\phi}^{\alpha\phi} := \min\{z_{\phi}^{\alpha\phi}(d) \mid d = 0,\ldots,C(\alpha,\phi)\}$

If $G^* > z_{\phi}^{\alpha\phi}$, **Then** $G^* := z_{\phi}^{\alpha\phi}$.

Step 4. Backtrack y_j^* to obtain the values of x_s^* and z_s^*.

Theorem 4.1 *Algorithm SPO terminates with the optimal solution to* **Single_ISDP** *in* $O(S\log S + CSJ^2)$ *time.*

Proof. We first prove that Algorithm **SPO** terminates at the optimal solution to **Single_ISDP**. For any given pair of $\alpha \in \{1,2,\cdots,S\}$ and $\phi \in \{1,2,...,J\}$, let $z_j^{\alpha\phi}(d)$ denote the minimum total cost when a subset of customers, $D \subseteq \{1,2,\cdots,j\}$, with $\sum_{k\in D}\lambda_k \le d$, is selected, where $j \in \{1,2,\cdots,\phi\}$. The algorithm correctly computes the values of $z_j^{\alpha\phi}(d)$ using the formula (4.14) and enumerates all the subsets $D \subseteq \{1,2,\cdots,\phi\}$ except for those that are either infeasible or dominated by the others for any given $\alpha \in \{1,2,\cdots,S\}$ and $\phi \in \{1,2,\cdots,J\}$. Thus, given (α,ϕ), the proposed algorithm identifies the optimal subsets of suppliers and customers subject to the revised capacity $C(\alpha,\phi)$. Furthermore, throughout the iteration process, **SPO** finds the optimal solution, $\{x_s^*\}$ and $\{y_j^*\}$, by enumerating all possible combinations of (α,ϕ) and solving $P^{\alpha\phi}$ for $\alpha = \arg\max_{s\in S}\{x_s^* > 0\}$ and $\phi = \arg\max_{j\in J}\{y_j^* = 1\}$. If the optimal solution utilizes only a zero capacity, then this solution by default is specified in the boundary condition. Otherwise, in the optimal solution, $\arg\max_{s\in S}\{x_s^* > 0\}$ and $\arg\max_{j\in J}\{y_j^* = 1\}$ are well defined. Therefore, Algorithm **SPO** solves **Single_ISDP** optimally.

We now show that the complexity of **SPO** is $O(S\log S + CSJ^2)$. Sorting suppliers, and customers, require $O(S\log S)$, and $O(J\log J)$, time in Step 1, respectively. The computational effort in calculating $G^{\alpha}(d)$ is $O(S\log S + CS)$ for all $\alpha \in \{1,2,\cdots,S\}$ and all $d = 0,1,2,\cdots,\min\left\{C, \min_{h\in \mathrm{H}}\left\{\sum_{s\in \mathrm{S}_h(\alpha)} F_s\right\}\right\}$ in Step 2. In Step 3, setting two types of boundary values takes $O(CSJ)$ and $O(SJ^2)$, respectively. In Step 4, for any given $\alpha \in \{1,2,\cdots,S\}$ and $\phi \in \{1,2,\cdots,J\}$, **SPO** finds $z_{\phi}^{\alpha\phi}$ in $O(CJ)$. Therefore, the time complexity of Step 4 is $O(CSJ^2)$, and therefore Algorithm **SPO** terminates in $O(S\log S + CSJ^2)$. □

Remark 1. Algorithm **SPO** may be applied to a more general setting where different customer orders may require different unit processing times (i.e., unit processing time τ_j for customer *j* at the PC). In this case, we can use

$$C(\alpha,\phi) = \min\left\{C, \min_{h\in \mathrm{H}}\left\{\sum_{s\in \mathrm{S}_h(\alpha)} F_s\right\}, \sum_{j=1}^{\phi}\lambda_j,\right\}$$

as the revised capacity, while keep $T_{\phi} - t_{\phi}^{DP} - t_{\alpha}^{\sup}$ as the maximum processing time at the PC, and generalize the recursive equation with the given (α,ϕ) as follows

$$z_j^{\alpha\phi}(d,u) = \begin{cases} z_{j-1}^{\alpha\phi}(d,u) & \text{if } d < \lambda_j \text{ or } u < \tau_j\lambda_j \\ \min\begin{cases} z_{j-1}^{\alpha\phi}(d,u) \\ \\ z_{j-1}^{\alpha\phi}(d-\lambda_j, u-\tau_j\lambda_j) + (a_j\lambda_j + \pi_j - p_j\lambda_j) + (G^{\alpha}(d) - G^{\alpha}(d-\lambda_j)) \\ +\gamma\cdot\{[\max\{0,\ \beta\cdot\sum_{k=1}^{J}\lambda_k - d\}]^{\omega} - [\max\{0,\ \beta\cdot\sum_{k=1}^{J}\lambda_k - d + \lambda_j\}]^{\omega}\} \end{cases} & \text{otherwise} \end{cases}$$

for $1 \le j \le \phi$, $0 \le d \le \min\left\{C, \min_{h\in \mathrm{H}}\left\{\sum_{s\in \mathrm{S}_h(\alpha)} F_s\right\}, \sum_{j=1}^{\phi}\lambda_j,\right\}$ and $0 \le u \le T_\phi - t_\phi^{DP} - t_\alpha^{\sup}$ where u denotes the remaining time from $T_\phi - t_\phi^{DP} - t_\alpha^{\sup}$. This generalization could result in another pseudo-polynomial time algorithm with, however, a higher level of computational complexity $O(S\log S + \max_{\forall j}\{T_j\}CSJ^2)$.

Remark 2. Algorithm **SPO** may also be extended to the case where the component-product conversion ratio becomes ρ_h:1, instead of 1:1. That is, to produce one unit of final product, we need ρ_h units of component h, *where* $\rho_h \ge 1$. When this is the case, we shall change equation (4.3) to $\sum_{s\in \mathrm{S}_h(\alpha)} x_s = \rho_h \cdot \sum_{j=1}^{\phi}\lambda_j y_j$ and then apply the same algorithm. However, if the component conversion ratios become customer-dependent, then we will no longer have a polynomial time solution. In this case, our problem becomes a multidimensional knapsack problem which is a strongly NP-hard problem.

4.3 Numerical Example

We now present a numerical example that illustrates the step-by-step search process by the proposed Algorithm **SPO** to find the optimal solution. In this example, we are given six suppliers for three types of supplies (S = 6, H = 3) and five customer demand points (J = 5), each has an order defined by (λ_j, T_j), as described in Tables 4.1-4.2. Also, we assume that for the PC, we have $\tau = 2$ and C = 8. Let the target fill rate β=87.5%, and the penalty cost parameter $\gamma = 26 \ge \max_{j\in \mathrm{J}}\{p_j\}$ and the penalty exponent ω =1.25.

Table 4.1 Parameters for the suppliers

Supplier s	Supplier set S_h ($s \in S_h$)	Capacity F_s	Shipping time $t_s^{\sup}$	Unit shipping cost b_s
1	S_1	5	1	3
2	S_2	8	1	3
3	S_3	7	2	4
4	S_1	5	3	2
5	S_3	3	4	2
6	S_2	4	6	1

Table 4.2 Parameters for the customer demand points

Customer j	Demand λ_j	Deadline T_j	Shipping time t_j^{DP}	Fixed cost π_j	Shipping cost a_j	Penalty cost p_j
	2	32	3	8	5	22
2	2	28	5	9	3	18
3	1	26	4	7	4	26
4	2	25	5	5	2	20
5	1	21	1	10	6	25

For α = 1 or 2, the revised capacity is always zero since all distinct three types cannot be provided (H = 3), i.e., $C(\alpha,\beta)$ = 0 for α = 1 or 2, and thus we summarize the steps for $\alpha = 3,4,5$ and 6. Table 3 shows the procedures for Step 2 and Tables 4.4-4.7 show the procedures for Step 4.

Table 4.3 The values of $S^{\alpha}(d)$ obtained in Step 2 of SPO

α	$\min\left\{C, \min\limits_{h\in H}\left\{\sum\limits_{s\in S_h(\alpha)} F_s\right\}\right\}$	d								
		0	1	2	3	4	5	6	7	8
3	5	0	10	20	30	40	50			
4	7	0	9	18	27	36	45	55	65	
5	8	0	7	14	21	30	39	49	59	69
6	8	0	5	10	15	22	31	41	51	61

Table 4.4 The values of $z_j^{\alpha\phi}(d)$ and $z_{\phi}^{\alpha\phi}$ obtained in Step 4 of **SPO** ($\alpha = 3$)

ϕ	$C(\alpha,\phi)$	j	d						$z_{\phi}^{\alpha\phi}$
			0	1	2	3	4	5	
1	2	0	467.0	467.0	467.0				359.4
		1	467.0	467.0	**359.4**				

2	4	0	467.0	467.0	467.0	467.0	467.0		266.7
		1	467.0	467.0	359.4	364.0	369.3		
		2	467.0	467.0	359.4	364.0	**266.7**		
3	5	0	467.0	467.0	467.0	467.0	467.0	467.0	220.8
		1	467.0	467.0	359.4	364.0	369.3	375.8	
		2	467.0	467.0	359.4	364.0	266.7	277.7	
		3	467.0	410.2	359.4	307.1	266.7	**220.8**	
4	5	0	467.0	467.0	467.0	467.0	467.0	467.0	210.8
		1	467.0	467.0	359.4	364.0	369.3	375.8	
		2	467.0	467.0	359.4	364.0	266.7	277.7	
		3	467.0	410.2	359.4	307.1	266.7	220.8	
		4	467.0	410.2	354.4	302.1	256.7	**210.8**	
5	5	0	467.0	467.0	467.0	467.0	467.0	467.0	210.8
		1	467.0	467.0	359.4	364.0	369.3	375.8	
		2	467.0	467.0	359.4	364.0	266.7	277.7	
		3	467.0	410.2	359.4	307.1	266.7	220.8	
		4	467.0	410.2	354.4	302.1	256.7	210.8	
		5	467.0	410.2	354.4	302.1	256.7	**210.8**	

Table 4.5 The value of $z_j^{\alpha\phi}(d)$ and $z_\phi^{\alpha\phi}$ obtained in Step 4 of **SPO** ($\alpha = 4$)

ϕ	$C(\alpha,\phi)$	j	d								$z_\phi^{\alpha\phi}$
			0	1	2	3	4	5	6	7	
1	2	0	467.0	467.0	467.0						357.4
		1	467.0	467.0	**357.4**						
2	4	0	467.0	467.0	467.0	467.0	467.0				262.7
		1	467.0	467.0	357.4	362.0	367.3				
		2	467.0	467.0	357.4	362.0	**262.7**				
3	5	0	467.0	467.0	467.0	467.0	467.0	467.0			215.8
		1	467.0	467.0	357.4	362.0	367.3	373.8			
		2	467.0	467.0	357.4	362.0	262.7	273.7			
		3	467.0	409.2	357.4	304.1	262.7	**215.8**			
4	7	0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	143.0
		1	467.0	467.0	357.4	362.0	367.3	373.8	383.4	399.2	
		2	467.0	467.0	357.4	362.0	262.7	273.7	288.6	311.0	
		3	467.0	409.2	357.4	304.1	262.7	215.8	232.9	257.6	
		4	467.0	409.2	352.4	299.1	252.7	205.8	174.0	**143.0**	
5	7	0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	143.0
		1	467.0	467.0	357.4	362.0	367.3	373.8	383.4	399.2	
		2	467.0	467.0	357.4	362.0	262.7	273.7	288.6	311.0	
		3	467.0	409.2	357.4	304.1	262.7	215.8	232.9	257.6	
		4	467.0	409.2	352.4	299.1	252.7	205.8	174.0	143.0	
		5	467.0	409.2	352.4	299.1	252.7	205.8	171.0	**143.0**	

Table 4.6 The value of $z_j^{\alpha\phi}(d)$ and $z_\phi^{\alpha\phi}$ obtained in Step 4 of **SPO** ($\alpha = 5$)

ϕ	$C(\alpha,\phi)$	j	d									$z_{\phi}^{\alpha\phi}$
			0	1	2	3	4	5	6	7	8	
1	2	0	467.0	467.0	467.0							353.4
		1	467.0	467.0	**353.4**							
2	4	0	467.0	467.0	467.0	467.0	467.0					256.7
		1	467.0	467.0	353.4	358.0	365.3					
		2	467.0	467.0	353.4	358.0	**256.7**					
3	5	0	467.0	467.0	467.0	467.0	467.0	467.0				209.8
		1	467.0	467.0	353.4	358.0	365.3	373.8				
		2	467.0	467.0	353.4	358.0	256.7	269.7				
		3	467.0	407.2	353.4	298.1	256.7	**209.8**				
4	7	0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0		137.0
		1	467.0	467.0	353.4	358.0	365.3	373.8	383.4	399.2		
		2	467.0	467.0	353.4	358.0	256.7	269.7	286.6	311.0		
		3	467.0	407.2	353.4	298.1	256.7	209.8	228.9	255.6		
		4	467.0	407.2	348.4	293.1	246.7	199.8	168.0	**137.0**		
5	8	0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	137.0
		1	467.0	467.0	353.4	358.0	365.3	373.8	383.4	399.2	435.0	
		2	467.0	467.0	353.4	358.0	256.7	269.7	286.6	311.0	356.4	
		3	467.0	407.2	353.4	298.1	256.7	209.8	228.9	255.6	306.0	
		4	467.0	407.2	348.4	293.1	246.7	199.8	168.0	137.0	191.9	
		5	467.0	407.2	348.4	293.1	246.7	199.8	165.0	**137.0**	138.0	

Table 4.7 The values of $z_j^{\alpha\phi}(d)$ and $z_\phi^{\alpha\phi}$ obtained in Step 4 of **SPO** ($\alpha = 6$)

ϕ	$C(\alpha,\phi)$	j	d								$z_{\phi}^{\alpha\phi}$
			0	1	2	3	4	5	6	7	
1	2	0	467.0	467.0	467.0						349.4
		1	467.0	467.0	**349.4**						
2	4	0	467.0	467.0	467.0	467.0	467.0				248.7
		1	467.0	467.0	349.4	354.0	361.3				
		2	467.0	467.0	349.4	354.0	**248.7**				
3	5	0	467.0	467.0	467.0	467.0	467.0	467.0			201.8
		1	467.0	467.0	349.4	354.0	361.3	371.8			
		2	467.0	467.0	349.4	354.0	248.7	263.7			
		3	467.0	405.2	349.4	292.1	248.7	**201.8**			
4	7	0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	129.0
		1	467.0	467.0	349.4	354.0	361.3	371.8	383.4	399.2	
		2	467.0	467.0	349.4	354.0	248.7	263.7	282.6	309.0	
		3	467.0	405.2	349.4	292.1	248.7	201.8	217.9	251.6	
		4	467.0	405.2	344.4	287.1	238.7	191.8	160.0	**129.0**	
5	7	0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	467.0	129.0
		1	467.0	467.0	349.4	354.0	361.3	371.8	383.4	399.2	
		2	467.0	467.0	349.4	354.0	248.7	263.7	282.6	309.0	

3	467.0	405.2	349.4	292.1	248.7	201.8	217.9	251.6
4	467.0	405.2	344.4	287.1	238.7	191.8	160.0	129.0
5	467.0	405.2	344.4	287.1	238.7	191.8	157.0	**129.0**

We then backtrack to find the optimal solution: $y_1^* = y_2^* = y_3^* = y_4^* = 1$, $y_5^* = 0$; $x_1^* = 2$, $x_2^* = 3$, $x_3^* = 4$, $x_4^* = 5$, $x_5^* = 3$, and $x_6^* = 4$; $z_1^* = 1$, $z_2^* = 1$, $z_3^* = 1$, $z_4^* = 1$, $z_5^* = 1$, and $z_6^* = 1$ with $(\alpha, \phi) = (6, 4)$, the utilized PC capacity of 7, and $G^* = 129.0$.

5 Linear Partial Relaxation-based Heuristic Algorithm

In this chapter, we study a general version of **ISDP** where multiple processing centers are considered. We provide some theoretical results of the linear relaxation problem of **ISDP**, upon which the design of the proposed algorithm for solving **ISDP** is based. Then we develop an iteration algorithm, partial linear relaxation-based heuristic algorithm, in which a small number of relaxed linear variables are fixed to be binary variables during each iteration. The iteration process stops until all the values of the binary variables are determined.

5.1 Heuristic Algorithm

In this section, we propose a LP relaxation based heuristic for solving **ISDP**. In the literature, there are many linear programming relaxation based algorithms for solving multiple knapsack problems or generalized assignment problems. Dawande et al. (2000) studied the multiple knapsack problems with the assignment restrictions and presented approximate solutions in polynomial computational time. Dahl et al. (2004) proposed a linear programming based heuristics for the multiple knapsack problems with assignment restrictions. Trick (1992) investigated the basis structure of the linear relaxation of the generalized assignment problems and presented an improvement heuristic based on the problem properties and violated inequalities. Our solution approach extends the existing results to solve a three-stage shipping network operations scheduling problem with delivery deadlines, supplier selections and customer assignments.

To begin, let us consider the following mixed integer linear program, **ISDP** $^{\mathbf{LR}}$, formed by relaxing binary variables y_{nj}, $y_{nj} \in \{0,1\}$ to bounded linear variables δ_{nj}, $0 \le \delta_{nj} \le 1$, , $\forall n \in N$, $\forall j \in J$.

$$\mathbf{ISDP^{LR}}: \quad \min \quad \sum_{s\in S}\sum_{n\in N} b_{sn}x_{sn} + \sum_{n\in N}\sum_{j\in J}\left(a_{snj}\lambda_{nj} + \pi_{nj}\right)\cdot\delta_{nj} + \sum_{j\in J} p_j\lambda_j(1-\sum_{n\in N}\delta_{nj})$$

$s.t.$

$$\sum\nolimits_{n\in N} x_{sn} \le F_s, s \in S \tag{5.1}$$

$$x_{sn} \le z_{sn}F_s, s \in S, n \in N \tag{5.2}$$

$$\sum\nolimits_{s\in S} x_{sn} = \sum\nolimits_{j\in J} \beta_{nj}\lambda_j\delta_{nj}, n \in N \tag{5.3}$$

$$\max\{t_{sn}z_{sn}\} + \sum_{s\in S} x_{sn}\tau_n + t_{nj}\delta_{nj} - M(1-\delta_{nj}) \le T_j, \quad j \in J \tag{5.4}$$

$$\sum\nolimits_{n\in N} \delta_{nj} \le 1, j \in J \tag{5.5}$$

$$\sum\nolimits_{j\in J} \lambda_j\delta_{nj} \le C_n, n \in N \tag{5.6}$$

$$x_{sn} \ge 0, \; z_{sn} \in \{0,1\}, \; 0 \le \delta_{nj} \le 1, \; s \in S, n \in N, j \in J \tag{5.7}$$

Since the number of customers, *J*, is usually much greater than the number of processing centers, *N*, and/or the number of suppliers, ***S***, such a partially relaxed model has potential to achieve a significant reduction in the number of integer variables. One remaining issue here is then how this relaxation may affect the solution quality. The analysis below serves for this purpose.

Let $\omega = (x, \delta) \in C$ be a feasible solution to **ISDP LR**, where δ denote the sub-vector of ω containing fractional variables ($0 < \delta_{nj} < 1$) and C denote the set of feasible solutions to **ISDP LR**. Let G^F be the sub-graph of G^{LR} induced by the fractional edges associated with δ_{nj}, where $G^{LR} = (V^{LR}, E^{LR})$, $V^{LR} = S \cup N \cup J$ and $E^{LR} = \{(s,n),(n,j) \mid \omega = (x_{sn}, \delta_{nj}) \in C\}$, $s \in S$, $n \in N$, $j \in J$. In the following theorem, we introduce the solution properties of **ISDP LR** that facilitates the design of our hybrid heuristic algorithm.

Theorem 5.1 *If* $\min_{\forall n \in N} C_n \geq \max_{\forall j \in J} \lambda_j$ *, then there exists a basic solution to* **ISDPLR**, ω *, in which all of the following properties hold:*

a) G^F *consists of trees* $W_1, W_2, \ldots, W_\gamma$, γ *denotes the number of trees in* G^F *;*

b) W_k *includes at most one customer demand point in* J *as a leaf node,* $1 \leq k \leq \gamma$ *;*

c) W_k *has at most one* PC_n, $n \in N$, *that is not used up to its maximum capacity* C_n *; or* W_k *has at most one demand point not assigned completely,* $1 \leq k \leq \gamma$ *;*

d) *If* W_k *has a demand point as a leaf node in* J *, then the corresponding inequality in (11) is strict,* $1 \le k \le m$.

Proof. Let W be any tree in G^F and we shall start with the proof of *a).*

a) To prove that a) holds, we need to show that G^F has no cycle. Hence, we suppose that G^F contains a cycle and the cycle incudes a node sequence from i to j, where $i, j \in V^F$. Letting l be the perturbation of a feasible solution to **ISDP^LR^**, now we can find a path from i to j, i.e.,

$$l = (+\varepsilon, (-\lambda_i/\lambda_{i+1})\varepsilon, (+\lambda_i/\lambda_{i+1})\varepsilon, (-\lambda_i/\lambda_{i+2})\varepsilon \ldots, (-\lambda_i/\lambda_{j-1})\varepsilon, (+\lambda_i/\lambda_j)\varepsilon)$$

with zero for other arcs (Dahl et al., 2004). Here we choose $\varepsilon > 0$ to be small enough, and construct two feasible solutions $v_1 = \omega + l$ and $v_2 = \omega - l$. We conclude that $\omega = 1/2(v_1 + v_2)$, which contradicts the fact that ω is the basic feasible solution.

b) By the similar arguments as in the proof of *a)*, consider that two demand points as leaf nodes in W_k , i.e., i and j. Define l to be the perturbation of the feasible solution to **ISDP^LR^** along the path from i to j in G^F. Let ε be positive and small enough, and then we have the following:

$$l = (+\varepsilon, (-\lambda_i/\lambda_{i+1})\varepsilon, (+\lambda_i/\lambda_{i+1})\varepsilon, (-\lambda_i/\lambda_{i+2})\varepsilon \ldots, (-\lambda_i/\lambda_{j-1})\varepsilon, (+\lambda_i/\lambda_j)\varepsilon).$$

We also construct two feasible solutions, $v_1 = \omega + l$ and $v_2 = \omega - l$, so that we have the same result as in the proof of *a),* $\omega = 1/2(v_1 + v_2)$, which is a contradiction against the extreme point ω.

c) Since W_k has $|V(W_k)|-|V(W_k)\cap S|-1$ edges associated with δ_{nj} where $V(W_k)\cap S$ denotes the set of supplier nodes in W_k, and ω is the basic solution to **ISDP$^{\mathbf{LR}}$,** at least $|V(W_k)|-|V(W_k)\cap S|-1$ inequalities from (11) and (12) must be active (Dahl et al., 2004), that is, they are equalities. Therefore, at most one PC is not used to its capacity or at most one demand point is not assigned completely in W_k.

d) If W_k *has a demand point* j *as a leaf node in* J, then only $\delta_{nj} > 0$ is fractional, thus $\sum_{n\in\mathbf{N}}\delta_{nj} < 1$. □

Theorem 5.2 *If* $\min_{\forall n\in N} C_n \geq \max_{\forall j\in J} \lambda_j$ *, then the number of demand points that are partially fulfilled, i.e.,* $0<\delta_{nj}<1$*, is no more than the number of PCs that offer supplies to partially assigned demand points, i.e.,* $\sum_{\forall k}\left|V(T_k)\cap N\right| \geq \sum_{\forall k}\left|V(T_k)\cap J\right|$.

Proof. Let W_k be any tree in G^F. Let r_k be the number of supplier nodes, p_k be the number of PC nodes, and q_k be the number of customer nodes in W_k. The number of edges in W_k is $r_k + p_k + q_k - 1$. Let θ_1, and θ_2, be the number of suppliers in $W_k \cap S$, and the number of customer demand points (leaf) in $W_k \cap J$, respectively. It follows that $\theta_2 \leq 1$from part *c)* of Theorem 1. Let d_i denote the degree of node i in W_k. Then, we have that the following inequality holds:

$$
\begin{aligned}
r_k + p_k + q_k - 1 = \sum\nolimits_{i\in W_k} d_i &= \theta_1 + \theta_2 + \sum\nolimits_{i\in W_k\cap S:\, d_i\geq 2} d_i + \sum\nolimits_{i\in W_k\cap J:\, d_i\geq 2} d_i \\
&\geq \theta_1 + \theta_2 + 2(r_k - \theta_1) + 2(q_k - \theta_2) \\
&\geq 2(r_k + q_k) - \theta_1 - \theta_2
\end{aligned}
$$

From the inequality above, it follows that $p_k - (r_k - \theta_1) \geq q_k \Rightarrow p_k \geq q_k$, which indicates that $\sum_{\forall k} |V(T_k) \cap N| \geq \sum_{\forall k} |V(T_k) \cap J|$ by summing up over all the trees, W_k, in G^F. □

Theorem 5.3 *There exists at least* $J - N$ *demand points that are fully fulfilled in the optimal solution to* ISDP^{LR}, i.e., $\delta_{nj} = 1$.

Proof. Let ω be the optimal solution to **ISDP^{LR}**. If G^F is empty, then there exists no customer demand point that is partially fulfilled by ω. In other words, customer demand points are either fully fulfilled by a single PC (i.e., the single coursing constraints (3)) or entirely unassigned. Let p be the number of PCs that serve partially fulfilled demand points in ω, α be the number of integrally fulfilled demand points in ω, η be the number of partially fulfilled demand points in ω. Hence $\alpha + \eta = J$ and we have that $\eta \leq p \leq N$ by Theorem 5.2. Since each demand point that is partially fulfilled is assigned to at least two PCs, we derive the following inequality, $\alpha + 2\eta \leq N + J$, based on the statement of *c)* in Theorem 5.1. This means that $\alpha \geq J - N$. □

According to the single-sourcing constraints (2.4), all such partial fulfilled demand point j with $0 < \delta_{nj} < 1$ must be reevaluated to be either $\delta_{nj} = 0$ or $\delta_{nj} = 1$ in order to obtain the final feasible solution. Nevertheless, Theorem 5.2 indicates that the total number of bounded linear variables with fractional values under the optimal solution to **ISDP^{LR}**, $0 < \delta_{nj} < 1$, is likely to be small comparing to |J|. This observation leads to our two-phase LP relaxation based heuristic, called **LPR**, for solving **ISDP**, where phase 2 is an iterative process and in each iteration a subset of variables with fractional values are fixed. To do so, let us define the *contribution* of demand point j if j is fulfilled by PC_n

$$r_{nj} = p_j\lambda_j - a_{nj}\lambda_j - \pi_{nj} \text{ for all } n \in N \text{ and } j \in J, \qquad (5.8)$$

and then this proposed heuristic is summarized in the table below.

Algorithm LW

Phase 1: Determine an initial assignment of demand points to PCs

Solve $\mathbf{ISDP}^{\mathbf{LR}}$, and fix $y_{nj} = 1$ if $\delta^*_{nj} = 1$, and let Δ is a pre-specified threshold. Note that for any link (n, j) over the shipping network, $\delta^*_{nj} = 1$ defines a feasible assignment that satisfies the PC capacities and delivery deadline constraints. Define

$$\Omega_0 = \{(n, j) | \delta^*_{nj} = 1\} \text{ and } \Omega = \{(n, j) | \text{ either } \delta^*_{nj} = 0 \text{ or } 0 < \delta^*_{nj} < 1\}, \forall n \in N, j \in J.$$

Phase 2: Iteration for solution improvement

a) Calculate the saving factor for each link $(n, j) \in \Omega$,

$$r_{nj} = p_j\lambda_j - a_{nj}\lambda_j - \pi_{nj}, (n, j) \in \Omega.$$

b) Based on the values of $r_{nj}, (n, j) \in \Omega$, partition Ω into Ω_1 and Ω_2 where

$$\Omega_1 = \{(n, j) | r_{nj} \ge \Delta\} \text{ and } \Omega_2 = \{(n, j) | r_{nj} < \Delta\}$$

c) Solve the following $\mathbf{ISDP}^{\mathbf{LR}'}$ upon the links defined by the given Ω

$$\mathbf{ISDP}^{\mathbf{LR}'}: \min \quad G = \sum_{s\in S}\sum_{n\in N} b_{sn}x_{sn} + \sum_{n\in N}\sum_{j\in J}(a_{nj}\lambda_j + \pi_{nj}) \cdot \delta_{nj} + \sum_{j\in J} p_j\lambda_j(1 - \sum_{n\in N}\delta_{nj})$$

$s.t.$

$$\sum_{n\in N} x_{sn} \le F_s, \qquad \forall s \in S$$

$$x_{sn} \le z_{sn}F_s, \qquad \forall s \in S, n \in N$$

$$\sum_{s\in S} x_{sn} = \sum_{j\in J} \lambda_j\delta_{nj}, \qquad \forall n \in N$$

$$\sum_{n\in N} \delta_{nj} \le 1, \qquad \forall (n, j) \in \Omega$$

$$\max\{t_{sn}z_{sn}\} + \sum_{s\in S} x_{sn}\tau_n + t_{nj}\delta_{nj} - M(1 - \delta_{nj}) \le T_j, \quad \forall j \in J$$

$$\sum_{j\in J} \lambda_j\delta_{nj} \le C_n, \qquad \forall n \in N$$

$$0 \le \delta_{nj} \le 1, \qquad \forall (n,j) \in \Omega_2$$

$$\delta_{nj} \in \{0,1\}, \qquad \forall (n,j) \in \Omega_1$$

δ_{nj} are fixed as binary constants, $\forall (n,j) \in \Omega_0$

d) Let δ_{nj}^* be the optimal solution to **ISDP**$^{\mathbf{LR}'}$, then update

$$\Omega_0 \Leftarrow \Omega_0 \cup \{(n,j) \mid \delta_{nj}^* = 1\} \text{ and } \Omega \Leftarrow \Omega / \{(n,j) \mid \delta_{nj}^* = 1\}.$$

e) Terminate if the value of G no longer improves.

In Algorithm **LPR**, the phase I solves ISDP$^{\mathrm{LR}}$ to obtain an initial assignment of customer demand points to PCs by fixing $y_{nj} = 1$ if the optimal solution to ISDP$^{\mathrm{LR}}$, δ_{nj} , equals to 1. However, customer demand points associated with $\delta_{nj}^* = 0$ or $0 < \delta_{nj}^* < 1$ may still be fully fulfilled at the optimal solution to ISDP. Hence, in Phase II, we divide set Ω into two disjoint subsets, Ω_1 and Ω_2, based on the value of saving factors, r_{nj} , defined by (5.8). Since customer demand points with a more significant saving if served are more likely to be fully fulfilled, we set variables, δ_{nj}, with higher saving factors to be binary variable. According to Theorem 5.2 that the number of variables with fractional value, δ_{nj} , is small, we can solve ISDP$^{\mathrm{LR}'}$ quickly. Theorem 5.3 guarantees that Algorithm LPR will fix some variables, y_{nj} , to be 1 in each iteration and the total number of binary valuables { y_{nj} } is bounded by a constant, the heuristic terminates within polynomial steps.

5.2 Computational Results

In this section, we report on the computational performance of the proposed search algorithm in the experiments defined by following parameters:

a) *The level of variability in customer demand, σ/μ*, where σ, and μ, stands for the standard deviation, and the mean, of the order sizes $\{\lambda_j\}$, respectively;

b) *The relative penalty cost, R*, defined by

$$R = \sum_{j\in J} p_j\lambda_j \Big/ \sum_{n\in N}\sum_{j\in J}(a_{nj}\lambda_j + \pi_{nj}) \tag{5.9}$$

where the value of parameter R increases as the level of shortage cost.

c) *The size of the distribution network*, |J|. In our experiment, the size of the network ranged from |J|=5 to |J|=65. For all the test cases with |J|>65, we were unable to obtain the optimal solution to problem CNOS within 30 minutes of CPU time (on a Dell 600, Pentium-M 1.4 GHz with 1 GB RAM). The main reason is that the CNOS problems we study in this work contains the generalized assignment problem as a sub-problem, which introduces the combinatorial nature into the search process and therefore the time needed to verify the optimality of a solution becomes excessive when the network size becomes large.

Table 5.1 Experimental design

Parameters	*Range*	Settings of the other parameters
J	5, 8, 12, 20, 50, 65, 80, 90, 100, 110, 120, 130, 140, 150	S=8, N=5
R	100%, 110%,120%,130%,140%,150%	$S\in\{5, 8\}$, $N\in\{3, 5\}$, $J\in\{5, 10\}$
σ/μ	10%, 15%, 20%, 25%, 30%, 35%, 40%	$S\in\{5, 8\}$, $N\in\{3, 5\}$, $J\in\{5, 10\}$

Table 5.2 Auxiliary parameters used in the experiments

Parameter	C_s	b_{sn}	t_{sn}	τ_n	a_{nj}	π_{nj}	t_{nj}	λ_j	T_j
Value	[100,1000]	[0.3,0.8]	[15,40]	[15,25]	[0.1,0.5]	[20,100]	[5,10]	[80,800]	[10,100]

For each given set of parameter values (|J|, R, σ/μ) in Table 5.1, 30 random test cases were generated for our empirical study. The computational performance of the proposed search algorithm was measured by its required CPU time (in second) on a Dell 600 (Pentium-M 1.4 GHz with 1 GB RAM) and the error gap defined below

$$\%\ Gap = \frac{G^{LPR} - G^*}{G^*} \qquad (5.10)$$

where G^* stands for the minimum operation cost obtained by using the commercial CPLEX solver to solve the respective CNOS problem defined by (2.1) - (2.7), and G^{LPR} stands for the total operation cost by our proposed two-phase search algorithm **LPR**.

In Tables 5.3-5.7, Columns 1-4 contain the following information in the order of network size (|**J**|), average CPU time required by CPLEX, the average running time of **LPR** algorithm, the average objective function value of CPLEX, and the average objective function value obtained by the **LPR** algorithm. Column 6 gives the average performance gap measured by (5.10). Each data point reported in these tables stands for the average of 30 observations from the randomly generated test cases.

Table 5.3 below reveals the impact of problem size in terms of the network size, *J*, on the algorithm performance. As we can see, when the number of demand points or network size is relatively small (*J*=5, 8, 12, 20, 50), using CPLEX solver directly to solve problem **CNOS** is a practical option. It requires only a minimal amount of CPU time while guarantees the optimality. However, as the problem size increases, CPLEX solver starts to lose its computational advantage to the proposed heuristic. Especially, when the problem size goes beyond J=50 in our experiments, the required computation time by the CPLEX solver becomes fairly excessive, while that required by the proposed algorithm **LPR** is constantly within 2 CPU seconds. The resulting error gaps are constantly within 3%, with most gaps within 2.5%. One main reason behind this observation is that with the proposed **LPR** heuristic, we solve a variation of the linear transshipment problem – a relatively easier problem, instead of the generalized assignment problems which is a much harder problem computationally.

Table 5.3 Impact of network size, |J|, on the algorithm performance

|J|	CPU Time (seconds)[1]		Performance in total cost[1]			
	CPLEX	LPR	CPLEX (G*)	LPR (G^{LPR})	Gap (%)	Std. Dev.
5	0.29271	0.037838	293052.7	294195.5	1.39%	0.02
8	2.730231	0.052462	66163.44	67589.51	2.22%	0.02
12	5.538677	0.052462	85544.04	87453.86	2.23%	0.01
20	8.329728	0.102374	93651.33	95750.96	2.24%	0.01
50	67.52745	0.16115	63401.65	64828.73	2.28%	0.01
65	164.3722	1.162197	66160.37	67631.21	2.29%	0.02
80	183.3276	2.33649	81525.77	83403.05	2.30%	0.01
90	222.9948	6.76009	79915.98	81702.53	2.35%	0.02
100	329.3213	12.07849	60608.96	62046.94	2.37%	0.02
110	466.752	19.69129	77604	79482.26	2.42%	0.01
120	639.8884	25.99809	57435.2	58840.46	2.45%	0.01
130	853.4773	40.39849	86308.32	88428.83	2.46%	0.02
140	2112.401	58.29209	86042.32	88224.45	2.54%	0.01
150	3421.671	71.07849	37921.84	38911.97	2.61%	0.01

[1] Each data listed here stands for the average of 30 observations.

In Tables 5.4-5.5, we present the results of an experiment in which we compare the performance of **LPR** algorithm against the levels of demand variability (σ/μ) with the following two sets of parameters: S = 5, N = 3, J = 5 and S = 8, N = 5, J = 10, respectively. As the results show, the proposed **LPR** algorithm works very nicely in seeking for the optimal solution under a more homogeneous demand. However, as the variability in demand increases, the level of error gaps also increases slightly. Indeed, we notice that the average gap increases from 1.40% to 2.2% as the variability level in demand increases from σ/μ = 10% to σ/μ =20%. Our second observation is that the proposed **LPR** algorithm is capable to handle the cases where the level of demand variation is large. As shown in Tables 4-5, when the value of σ/μ exceeds 25%, the average error gap has the tendency to increase at a fairly reasonable rate. For instance, when $\sigma/\mu > 25\%$ for the case with S = 5, N = 3, J = 5, the average error gap is about 1.99% and as for the cases with S = 8, N = 5, J = 10, the average error gap is about 2.56%.

Table 5.4 LPR algorithm vs. CPLEX on σ/ μ (S=5 N=3 J=5)

σ/μ (%)	CPU Time (seconds)[1]		Performance in total cost[1]			
	CPLEX	LPR	CPLEX (G*)	LPR (G^{LPR})	Gap (%)	Std. Dev.
10	0.304493	0.042883	868760.3	881201.8	1.43%	0.01
15	0.306397	0.03382	59555.73	60511.59	1.60%	0.01
20	0.295177	0.025183	59671.34	60793.98	1.88%	0.01
25	0.28583	0.030893	62322.72	63560.39	1.99%	0.02
30	0.347783	0.025607	59868.15	61114.19	2.08%	0.01
35	0.399228	0.027595	60544.49	61935.89	2.30%	0.02
40	0.412017	0.030187	41605.5	42640.01	2.49%	0.01

[1] Each data listed here stands for the average of 30 observations.

Table 5.5 LPR algorithm vs. CPLEX on σ/ μ (S=8 N=5 J=10)

σ/μ (%)	CPU Time (seconds)[1]		Performance in total cost[1]			
	CPLEX	LPR	CPLEX (G*)	LPR (G^{LPR})	Gap (%)	Std. Dev.
10	219.1416	3.4196	2393402	2432013	1.61%	0.01
15	231.0398	3.518	1971913	2009988	1.93%	0.02
20	240.9033	3.3141	343295	351039.8	2.26%	0.01
25	256.1943	3.6156	1943578	1988117	2.29%	0.02
30	258.1018	3.4829	1931199	1976402	2.34%	0.02
35	259.0081	3.5075	2318260	2375737	2.48%	0.01
40	268.7694	3.4831	1741057	1784030	2.47%	0.01

[1] Each data listed here stands for the average of 30 observations.

From Table 5.4 and Table 5.5, we can see that when the level of σ/μ is relatively low, the resulting empirical error gaps are fairly small. This indicates that the **LPR** algorithm has the potential to find near optimal solutions when the customer demands in a supply chain network are more homogeneous. This observation is consistent with the fact that a knapsack problem can be easily solved optimally if all the items (ordered) sizes are equal (i.e., σ/μ). However, after the level of σ/μ goes beyond 25%, the empirical error gaps tend to increase. Nevertheless, about 72% of empirical average error gaps were within 2.4% from the optimal values obtained in our experiment, with the largest average error gap 2.48%.

Table 5.6 LPR algorithm v.s. CPLEX on R (S=5 N=3 J=5)

R (%)	CPU Time (seconds)[1]		Performance in total cost[1]			
	CPLEX	LPR	CPLEX (G*)	LPR (G^{LPR})	Gap (%)	Std. Dev.
100	0.297637	0.024263	77346.87	79737.32	3.09%	0.02
110	0.291357	0.024130	84759.02	87394.06	3.11%	0.01
120	0.293400	0.024777	67180.79	68851.04	2.49%	0.01
130	0.296697	0.026237	63614.78	65358.51	2.74%	0.01
140	0.297017	0.026690	55917.41	57016.72	1.97%	0.01
150	0.304967	0.024990	58698.15	59810.35	1.89%	0.01

[1] Each data listed here stands for the average of 30 observations.

Table 5.7 LPR algorithm vs. CPLEX on R (S=8 N=5 J=10)

R (%)	CPU Time (seconds)[1]		Performance in total cost[1]			
	CPLEX	LPR	CPLEX (G*)	LPR (G^{LPR})	Gap (%)	Std. Dev.
100	7.3927	0.1100	64739.35	66535.97	2.78%	0.02
110	8.1032	0.1210	57703.48	59217.53	2.62%	0.02
120	8.2268	0.1141	77454.95	79168.09	2.21%	0.02
130	8.4321	0.1112	76506.89	78187.34	2.20%	0.01
140	8.6006	0.1129	62744.66	63870.97	1.80%	0.01
150	9.1991	0.1159	57379.4	58283.94	1.58%	0.01

[1] Each data listed here stands for the average of 30 observations.

In Tables 5.6-5.7, we report the impact on empirical error gaps by the relative penalty cost (R) as defined by (5.9). As the value of R goes beyond certain level, the total penalty cost dominates the shipping cost. According to the improvement criterion, the proposed **LPR** algorithm will now allocate the demand points to proper processing centers to lessen the penalty cost as much as possible. As we see in Tables 5.6-5.7, it appears that the average gap between the objective function values by the **LPR** algorithm and that by the CPLEX solver steadily decreases when the value of R increases from 100% to 150% for the two instances with S =5, N =3, J =5 and S =8, N =5, J =10, respectively.

6 Conclusion and Future Extensions

In this work, we studied the problem of scheduling the multi-echelon supply chain operations with customer delivery deadlines. To develop new methodologies for solving this problem, we analyzed three strongly polynomial time solvable cases of the scheduling problem, developed a dynamic programming-based algorithm for solving a special case with a single PC, and proposed a linear programming partial relaxation-based search algorithm to solve the general version of this problem. Meanwhile, we presented numerical examples and conducted empirical observations on the heuristic algorithm. The computational performances obtained from randomly generated test cases are reported. It is observed that the partial relaxation approach consistently obtained near optimal solution (within 2%) in 600 instances out of 800 test cases. There are several major extensions for this research.

1) Supply Chain Operations Scheduling with Multi-Modal Shipping Modes

This research was motivated by DHL SEAIR multimodal service. DHL SEAIR moves goods by ocean from Asia to a connecting transit hub in Dubai/Vancouver/Los Angeles, then transferring to flights into Europe, Middle East, Africa and Latin America. DHL SEAIR provides a service that is faster than pure ocean freight and more economical than standard air freight. This saves both cost - up to 50% versus standard air freight, and time, up to 50% versus standard ocean freight.

The multimodal transport modes will be introduced into the above operations scheduling problem. Mathematical properties of this problem will then be analyzed to find some special cases which can lead to polynomial-time optimal solutions. And then based on these efficient special cases, fast heuristic algorithms will be developed to solve the scheduling problem with multimodal shipping modes. Afterwards, computational experiments will be conducted to verify the effectiveness of the proposed algorithm.

2) The multimodal transshipment/distribution problem with convex cost functions

We shall allow multiple transportation modes along each network link from the source/supplier locations and customer demand points, and aim at choosing the optimal shipping mode and transshipment locations, each of which can also be a demand point, under convex cost functions. The fundamental properties of this problem will be analyzed, and the methodology for solving a generalized version of this problem will be explored. We shall prove that solving this optimization problem as a mixed integer program with non-linear cost functions is not much harder that its linear objective function version. Special cases of this problem will be analyzed and solution methodologies will be developed.

3) Supply Chain Operations with Uncertainty

Modern supply chains are very complex, and recent lean practices have resulted in these networks becoming more vulnerable. For instance, there is often little buffer inventory and any disruption can have a rapid impact on the supply process. Since 9/11, managers are more aware of the vulnerability of their supply chains, but most of them are still confused on the way to manage risk disruptions. First of all, this is simply because there is no easy answer in crafting strategies under uncertainty. Second, it is because managers are reluctant to implicate recent manufacturing practices; years of optimization theory have led to the supply chain as it is currently and including the notion of risks is a difficult challenge. Finally, as we will see, although managers do understand the costs that are related to risk management, they have a hard time quantifying the benefits.

With the Japan earthquake and tsunami and Thailand flooding, major manufacturers with global suppliers have had to halt operations. Can disaster planning really help prevent these supply chain disruptions? Are the risks associated with lean inventories worth the cost when operations are halted until supplies are re-stocked? Natural disasters like this, and even the current BP oil leak, can prompt manufacturers to look more closely at their master operating plans. Therefore, my future research will concentrate on the formulation and solution approach for the supply chain operations problems with uncertainty in the global supply chain.

Bibliography

Ahuja, R., Magnanti, T., & Orlin, J. (1993). *Network Flows: Theory, Algorithms, and Applications.* New York: Prentice Hall.

Alpan, G., Ladier, A., Larbi, R., & Penz, B. (2011). Heuristic solutions for transshipment problems in a multiple door cross docking warehouse. *Computers & Industrial Engineering, 61*(2), 402-408.

Amorima, P., Güntherb, H.-O., & B., A.-L. (2012). Multi-objective integrated production and distribution planning of perishable products. *International Journal of Production Economics, 138*(1), 89-101.

Bard, J., & NananuKul, N. (2009). Integrated production-inventory-distribution-routing problem. *Journal of Scheduling, 12*(3), 257-280.

Bashiri, M., Badri, H., & Talebi, J. (2012). A new approach to tactical and strategic planning in production–distribution networks. *Applied Mathematical Modeling, 36*(4), 1703-1717.

Bertazzi, L., & Zappa, O. (2012). Integrating transportation and production: an international study case. *Journal of the Operational Research Society, 63*, 920–930.

Bhutta, K.S. (2003). An integrated location, production, distribution and investment model for a multinational corporation. *International Journal of Production Economics, 86*(3), 201-216.

Brotcorne, L., Hanafi, S., & Mansi, R. (2009). A dynamic programming algorithm for the bi-level knapsack problem. *Operations Research Letters, 37*(3), 215-218.

Chandra, P., & Fisher, M. (1994). Coordination of Production and Distribution Planning. *European Journal of Operational Research, 72*(3), 503-517.

Chang, Y., & Lee, Y. (2004). Machine scheduling with job delivery coordination. *European Journal of Operational Research, 158*(2), 470-487.

Chen, Z., & Pundoor, G. (2006). Order assignment and scheduling in a supply chain. *Operations Research, 54*(3), 555-572.

Chen, Z., & Vairaktarakis, G. L. (2005). Integrated Scheduling of Productoin and Distribution Operations. *Management Science, 51*(4), 614-628.

Chiang, W.-C., Russell, R., Xu, X.-J., & Zepeda, D. (2009). A simulation/metaheuristic approach to newspaper production and distribution supply chain problems. *International Journal of Production Economics, 121*(2), 752-767.

Coffman, E. G., Jr., G. M., & Johnson, D. S. (1987). Bin-Packing with Divisible Item Sizes. *Journal of Complexity, 3*(4), 406-428.

Cohen, M., & Lee, H. (1988). Strategic Analysis of Integrated Production-Distribution Systems: Models and Methods. *Operations Research, 36*(2), 216-228.

Dahl, G., & Foldnes, N. (2006). LP based heuristics for the multiple knapsack problem with assignment restrictions. *Annuals of Operations Research, 146*(1), 91-104.

Dawande, M., Keskinocak, P., & Ravi, R. (2000). Approximation Algorithms for the Multiple Knapsack Problems with Assignment Restrictions. *Journal of Combinatorial Optimization, 4*(2), 171-186.

Detti, P. (2009). A polynomial algorithm for the multiple knapsack problem with divisible item sizes. *Information Processing Letters, 109*(11), 582-584.

Eksioglu, S. D., Eksiogulu, B., & Romeijn, H. E. (2007). A Lagrangean heuristic for integrated production and transportation planning problems in a dynamic, multi-item, two-layer supply chain. *IIE Transactions, 39*(2), 191-201.

Fumero, F., & Vercellis, C. (1999). Synchronized development of production, inventory and distribution schedules. *Transportation Science, 33*(3), 330-340.

Garey, M., & Johnson, D. (1979). *Computers and Intractability: A guide to the theory of NP-completeness.* Germany: W. H. Freeman.

Gaudreault, J., Forget, P., Frayret, J. M., Rousseau, A., Lemieux, S., & Amours, S. (2010). Distributed Operations Planning for the Softwood Lumber Supply Chain: Optimization and coordination. *International Journal of Industrial Engineering, 17*(3), 168-189.

Gebennini, E., Gamberini, R., & and Manzini, R. (2009). An integrated production¨Cdistribution model for the dynamic location and allocation problem with safety stock optimization. *International Journal of Production and Economics, 122*(1), 286-304.

Guinet, A. (2001). Multi-site planning: A transshipment problem. *International Journal of Production Economics, 74*(1), 21-32.

Hall, N., & Potts, C. (2005). The coordination of scheduling and batch deliveries. *Annals of Operations Research, 135*(1), 41-64.

Kannegiesser, M., & Günther, H.-O. (2011). An integrated optimization model for managing the global value chain of a chemical commodities manufacturer. *Journal of the Operational Research Society, 62*, 711–721.

Kellerer, H., & Pferschy, U. (2004). Improved dynamic programming in connection with an FPTAS for the knapsack problem. *Journal of Combinatorial Optimization, 8*(1), 5-11.

Kleinschmidt, P., & Schannath, H. (1995). A strongly polynomial algorithm for the transportation problem. *Mathematical Programming, 98*(1-3), 1-13.

Kogan, K., & Shtub, A. (1997). The Dynamic Generalized Assignment Problem. *Annuals of Operations Research, 69*(0), 227-239.

Korte, B., & Vyen, J. (2006). *Combinatorial Optimization: Theory and Algorithms.* New York: Springer-Verlag.

Lei, L., & Wang, G. (2012). Polynomial-time solvable cases of the capacitated multi-echelon shipping network scheduling problem with delivery deadlines. *International Journal of Production Economics, 137*(2), 263-271.

Lei, L., Liu, S., Ruszczynski, A., & Park, S. (2006). On the Integrated Production, Inventory, and Distribution Routing Problem. *IIE Transactions, 38*(11), 955-970.

Lei, L., Zhong, H., & Chaovalitwongse, W. (2009). On the integrated production and distribution problem with bidirectional flows. *INFORMS Journal on Computing, 21*(4), 585-598.

Lo, S., Wee, H., & Huang, W. (2007). An integrated production-inventory model with imperfect production processes and Weibull distribution deterioration under inflation. *International Journal of Production Economics, 106*(1), 240-260.

Miller, R. (1999). Optimization: Foundations and Applications. New York: Wiley-Interscience.

Rong, A., Akkerman, R., & and Grunow, M. (2011). An optimization approach for managing fresh food quality throught the supply chain. *International Journal of Production Economics, 131*(1), 421-429.

Sawik, T. (2009). Monolithic versus hierarchical approach to integrated scheduling in a supply chain. *International Journal of Production Research, 47*(21), 5881-5910.

Trick, M. A. (1992). A linear relaxation heuristic for the generalized assignment problem. *Naval Research Logistics, 39*(2), 137-151.

Wang, H., & Lee, C. (2005). Production and transport logistics scheduling with two transport mode choices. *Naval Research Logistics, 52*(8), 796-809.

Yan, C.-Y., Banerjee, A., & and Yang, L.-B. (2011). An integrated production¨Cdistribution model for a deteriorating inventory item. *International Journal of Production Economics, 133*(1), 228-232.

Yan, S., Juang, D., Chen, C., & Lai, W. (2005). Global and local search algorithms for concave cost transshipment problems. *Journal of Global Optimization, 33*(1), 123-156.

Zegordi, S. H., & Beheshti Nia, M. A. (2010). Integrating production and transportation scheduling in a two-stage supply chain considering order assignment.

International Journal of Advanced Manufacturing Technology, 44(9-10), 928-939.

Appendix A Computational Results of Algorithm LW

Table A.1 Impact of network size, |J|, on the algorithm performance

Run	J=5					J=8				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.2212	43288.44	0.0328	43288.44	0.00%	1.7312	88675.24	0.0551	90522.26	2.08%
2	0.2480	38034	0.0250	38164.3	0.34%	3.1216	46977.87	0.0525	47952.13	2.07%
3	0.3655	313330	0.0681	318330	1.60%	2.2197	74786.44	0.0614	75828.61	1.39%
4	0.3823	44330.3	0.0636	44330.6	0.00%	3.5301	45981.93	0.0490	47466.89	3.23%
5	0.3132	515000	0.0391	515210	0.04%	2.9563	75148.24	0.0574	76334.98	1.58%
6	0.3243	4005100	0.0208	4006012	0.02%	1.4310	67597.12	0.0437	68695.62	1.63%
7	0.2701	96060	0.0168	100240.3	4.35%	3.2776	53487.09	0.0470	53635.68	0.28%
8	0.2282	200000	0.0607	200128	0.06%	3.4229	57040.89	0.0587	57809.62	1.35%
9	0.2922	43330	0.0680	43330	0.00%	3.0428	40951.9	0.0626	42470.78	3.71%
10	0.3112	87860	0.0787	87860	0.00%	1.4169	89402.73	0.0531	91582.7	2.44%
11	0.334	93910	0.0317	93940.3	0.03%	2.5880	56251.12	0.0520	58228.75	3.52%
12	0.3155	46012	0.0421	47140.3	2.45%	3.3389	54207.99	0.0478	56447.19	4.13%
13	0.2630	30025	0.0195	30046	0.07%	3.8709	54978.79	0.0604	56106.78	2.05%
14	0.2434	15678	0.0156	16012.3	2.13%	3.6635	79737.97	0.0559	82872.9	3.93%
15	0.2611	111499.1	0.0132	116129.9	4.15%	1.7677	31394.35	0.0411	31736.04	1.09%
16	0.3290	45412	0.0124	47430	4.44%	3.3395	31898.81	0.0557	32640.44	2.32%
17	0.2986	200000	0.0409	200128	0.06%	1.5810	37921.84	0.0402	38911.97	2.61%
18	0.3294	43330	0.0223	43330	0.00%	2.6877	95661.63	0.0558	96633.08	1.02%
19	0.3042	57860	0.0362	60160	3.98%	2.3411	86308.32	0.0481	88428.83	2.46%
20	0.2312	82012	0.0146	83210	1.46%	3.4315	76945.2	0.0445	78572.3	2.11%
21	0.2813	42036.8	0.0718	43684.13	3.92%	2.5739	86079.15	0.0588	88525.78	2.84%
22	0.3648	50436.21	0.0252	52147.95	3.39%	1.5832	89554.01	0.0406	89832.02	0.31%
23	0.3355	59023.41	0.0262	60513.13	2.52%	2.6880	111753.7	0.0580	112110.7	0.32%
24	0.3310	56758.15	0.0228	57046.83	0.51%	2.3415	88547.69	0.0293	91887.3	3.77%
25	0.2826	57556.46	0.0220	57988.4	0.75%	3.4356	45813.8	0.0671	46900.46	2.37%
26	0.2286	49080.36	0.0244	50615.27	3.13%	2.5753	55052.41	0.0449	57259.76	4.01%
27	0.2768	66503.96	0.0201	67968.07	2.20%	1.5877	91749.3	0.0415	95451.56	4.04%

28	0.2947	71262.13	0.0225	71497.72	0.33%	2.6906	32915.52	0.0575	33592.17	2.06%
29	0.3136	39576.01	0.0291	40088.75	1.30%	2.3489	-11522.6	0.0361	-12024.3	4.35%
30	0.3140	34245.05	0.0287	35133.77	2.60%	3.4422	-2235.64	0.0640	-2241.33	0.25%

Table A.2 Impact of network size, |J|, on the algorithm performance (Con't)

Run	J=12					J=20				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	5.7835	57633.91	0.8163	57679.61	0.08%	7.05965	94364.84	0.114782	94673.39	0.33%
2	6.7767	46086.63	0.820121	46815.99	1.58%	8.153869	93485.06	0.113093	93540.89	0.06%
3	6.4816	74866.59	0.806836	75985.37	1.49%	9.917107	87424.5	0.093261	89553.14	2.43%
4	5.9714	44584.86	0.770038	45810.59	2.75%	8.97502	96927.91	0.10491	103154.1	6.42%
5	4.5680	93651.33	0.694366	95750.96	2.24%	9.517485	76655.66	0.092253	77182.01	0.69%
6	5.9788	36610.41	0.658025	38301.53	4.62%	7.067543	63426.5	0.095318	64823.59	2.20%
7	4.5319	68745.71	0.801731	69294.1	0.80%	8.131906	44514.16	0.091231	46519.19	4.50%
8	5.4089	69785.43	0.607282	73071.16	4.71%	7.97417	63958.53	0.092351	67004.9	4.76%
9	5.2612	48149.51	0.681081	50090.33	4.03%	10.10799	91835.72	0.116352	94079.5	2.44%
10	4.4336	32948.02	0.56709	33156.05	0.63%	10.42584	39044.49	0.092534	39494.5	1.15%
11	6.0516	73907.76	0.825689	76648.08	3.71%	6.916429	79592.9	0.092635	82206.09	3.28%
12	4.3986	85723.1	0.652684	86526.86	0.94%	7.198528	104023.9	0.094146	104061.8	0.04%
13	6.7839	83231.22	0.834934	83272.88	0.05%	7.348375	97364.63	0.114233	98319.08	0.98%
14	5.5044	55546.04	0.556478	56904.48	2.45%	10.45188	93546.03	0.104326	94600.36	1.13%
15	6.7359	78604.39	0.680317	79194.46	0.75%	8.184652	77058.2	0.100087	77139.17	0.11%
16	3.9218	86213.66	0.566952	86403.35	0.22%	7.83269	68772.84	0.115519	72104.91	4.85%
17	5.6904	79473.97	0.558958	80524	1.32%	7.192276	76335.41	0.111532	80227.65	5.10%
18	6.1637	80518.44	0.542859	84175.41	4.54%	8.172877	66612.7	0.093612	68625.51	3.02%
19	5.3709	33211.72	0.721482	34169.44	2.88%	8.041657	86042.32	0.106547	88024.45	2.30%
20	4.1955	64622.77	0.560652	66317.72	2.62%	8.165951	80374.3	0.11084	81823.62	1.80%
21	6.2989	95316.68	0.644345	99287.24	4.17%	8.088395	96875.06	0.100285	98595.39	1.78%
22	6.1929	35906.47	0.573921	37618.21	4.77%	7.201303	83750.8	0.105033	87556.71	4.54%
23	5.5560	41335.24	0.829872	42824.96	3.60%	8.18007	39316.06	0.115194	40475.38	2.95%
24	5.8251	38223.68	0.656155	38512.36	0.76%	8.043053	80492.89	0.113603	81888.01	1.73%
25	5.0198	38946.9	0.837788	39378.83	1.11%	8.167639	108422.1	0.102575	109305.2	0.81%

26	5.2586	34766.79	0.556595	36301.7	4.41%	8.089501	97171.2	0.109838	100357.7	3.28%
27	5.2609	46288.12	0.68322	47752.22	3.16%	7.202147	57407.66	0.110886	57468.23	0.11%
28	4.9560	47822.22	0.573852	48057.81	0.49%	8.181907	66946.82	0.091858	67253.46	0.46%
29	5.5574	27067.67	0.560387	27580.4	1.89%	8.04772	85487.73	0.110981	86164.3	0.79%
30	4.9266	24015	0.543661	24903.72	3.70%	8.173374	78900.86	0.118496	78993.39	0.12%

Table A.3 Impact of network size, |J|, on the algorithm performance (Con't)

Run	J=50					J=65				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	95.73663	52399.78	0.154758	52510.46	0.21%	997.0197	61099.53	1.707064	62927.61	2.99%
2	53.66383	97150.9	0.163765	97341.9	0.20%	989.2864	59767.74	2.204828	60327.54	0.94%
3	90.37819	66211.86	0.186468	70453.77	6.41%	972.9428	73870.65	0.325722	76980.01	4.21%
4	76.35046	70472.87	0.192202	70672.87	0.28%	978.1642	42679.76	0.516363	44970.45	5.37%
5	49.91873	81525.77	0.145619	83203.05	2.06%	972.9083	41427.33	2.173319	42960.86	3.70%
6	55.00465	48002.87	0.154509	50221.28	4.62%	943.6285	62648.81	1.220847	64752.41	3.36%
7	42.67561	69934.46	0.132921	71494.94	2.23%	927.1662	69946.16	0.970988	70669.21	1.03%
8	80.21126	56127.56	0.162995	56327.56	0.36%	943.7258	85544.04	1.390937	87453.86	2.23%
9	96.75856	49887.24	0.111146	51641.3	3.52%	912.1351	79762.92	1.559757	83467.73	4.64%
10	88.27884	78507.43	0.148644	81844.97	4.25%	965.4421	48382.58	0.554582	50486.66	4.35%
11	49.69013	43738.28	0.186585	46205.81	5.64%	960.6469	39405.76	1.79941	40880.77	3.74%
12	96.09661	55036.03	0.19859	56892.61	3.37%	936.6232	42956.83	0.977745	43520.62	1.31%
13	74.9037	47396.4	0.15705	47596.4	0.42%	990.9526	55480.35	0.297074	56473.59	1.79%
14	75.48693	59125.71	0.146738	59325.71	0.34%	930.1659	53251.22	1.496489	53596.36	0.65%
15	50.64124	56002.3	0.134966	56202.3	0.36%	996.6266	74831.91	1.133713	75408.15	0.77%
16	46.59066	59486.79	0.123041	59686.79	0.34%	956.1321	86609.29	1.745104	89261.41	3.06%
17	84.28092	85337.96	0.192802	85537.96	0.23%	969.5148	51932.74	0.303633	54225.89	4.42%
18	41.97382	91427.61	0.16417	94753.13	3.64%	975.493	57435.2	1.218493	58840.46	2.45%
19	38.62945	68517.56	0.182638	69368.49	1.24%	996.2512	43710.27	1.368731	44038.58	0.75%
20	48.39136	83611.3	0.189043	83906.26	0.35%	949.439	55221.25	0.249945	56036.31	1.48%
21	82.4149	69467	0.155493	75067.95	8.06%	987.551	84723.92	1.191401	85707.18	1.16%
22	81.16085	38474.08	0.116904	40185.82	4.44%	969.52	71719.83	1.119511	72900.77	1.65%
23	52.75776	43569.82	0.149121	45059.54	3.41%	975.499	31936.93	1.958757	32324.84	1.21%

24	53.25934	38656.7	0.186645	38945.38	0.74%	996.252	40254.61	0.302481	41553.55	3.23%
25	50.70062	39594.8	0.200867	40026.74	1.09%	949.4418	66303.05	1.414855	69421.61	4.70%
26	53.42963	37069.15	0.161137	38604.06	4.14%	987.5526	70780.64	1.101095	73282.58	3.53%
27	62.5951	48484.27	0.150547	49948.37	3.01%	969.5232	37504.92	0.180399	38173.66	1.78%
28	83.73083	48175.6	0.135759	48411.2	0.48%	975.5045	49129.94	1.068582	49951.8	1.67%
29	53.6129	27836.77	0.131728	28349.51	1.84%	996.2548	84581.5	1.099659	86339.37	2.08%
30	76.50614	25348.08	0.193083	26236.8	3.50%	949.4462	55109.6	1.927121	55662.65	1.00%

Table A.4 LW algorithm v.s. CPLEX on σ/ μ (S=5 N=3 J=5)

Run	σ/μ=10%					σ/μ=15%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.2212	43288.44	0.0328	43288.44	0.00%	0.2972	42142.04	0.0458	43228.91	2.58%
2	0.2480	38034	0.0250	38164.3	0.34%	0.3630	76217.36	0.0311	78502.67	3.00%
3	0.3655	313330	0.0681	318330	1.60%	0.2175	57366.89	0.0290	59986.83	4.57%
4	0.3823	44330.3	0.0636	44330.6	0.00%	0.2690	33565.37	0.0403	33782.14	0.65%
5	0.3132	515000	0.0391	515210	0.04%	0.2852	29991.6	0.0332	31475.72	4.95%
6	0.3243	4005100	0.0208	4006012	0.02%	0.2141	54635.49	0.0291	54725.64	0.17%
7	0.2701	96060	0.0169	100240.3	4.35%	0.3486	29835.38	0.0137	31338.72	5.04%
8	0.2282	200000	0.0608	200128	0.06%	0.2104	68039.24	0.0300	69007.24	1.42%
9	0.2922	43330	0.0680	43330	0.00%	0.2950	65773.55	0.0488	66674.21	1.37%
10	0.3112	87860	0.0787	87860	0.00%	0.2814	42953.16	0.0445	44980.26	4.72%
11	0.3340	93910	0.0318	93940.3	0.03%	0.2886	80090.79	0.0442	82784.69	3.36%
12	0.3155	46012	0.0422	47140.3	2.45%	0.3432	51494.02	0.0330	53053.42	3.03%
13	0.2630	30025	0.0196	30046	0.07%	0.2807	33735.31	0.0428	33746.25	0.03%
14	0.2434	15678	0.0156	16012.3	2.13%	0.3437	36451.03	0.0475	37213.76	2.09%
15	0.2611	111499.1	0.0132	116129.9	4.15%	0.2723	84898.31	0.0131	85428.39	0.62%
16	0.3290	45412	0.0124	47430	4.44%	0.3047	90159	0.0349	94298.67	4.59%
17	0.2986	200000	0.0409	200128	0.06%	0.3846	70672.35	0.0492	70943.88	0.38%
18	0.3294	43330	0.0223	43330	0.00%	0.2724	97228.59	0.0480	98225.76	1.03%
19	0.3042	57860	0.0362	60160	3.98%	0.4609	70489.77	0.0413	71795.43	1.85%
20	0.2312	82012	0.0146	83210	1.46%	0.2289	87293.38	0.0225	88697.35	1.61%
21	0.2813	42036.8	0.0718	43684.13	3.92%	0.2241	82511.99	0.0306	84885.94	2.88%

22	0.3609	3839426	0.0752	3846321	0.18%	0.4412	94142.13	0.0342	95761.78	1.72%
23	0.3109	3890217	0.0725	3995997	2.72%	0.3963	66808	0.0146	67995.24	1.78%
24	0.3666	1867479	0.0429	1899340	1.71%	0.3538	52462.59	0.0362	53046	1.11%
25	0.3000	3181271	0.0543	3182994	0.05%	0.2498	52964.02	0.0440	54328.19	2.58%
26	0.3407	1308196	0.0661	1350550	3.24%	0.2304	62529.31	0.0227	63278.28	1.20%
27	0.3025	540466.4	0.0337	540596.6	0.02%	0.4282	38321.22	0.0288	38741.72	1.10%
28	0.3351	1338248	0.0396	1374613	2.72%	0.2691	42790.44	0.0302	43600.51	1.89%
29	0.3119	2827348	0.0413	2937205	3.89%	0.2953	40059.18	0.0235	40884.43	2.06%
30	0.3593	1116051	0.0665	1130332	1.28%	0.3423	60423.06	0.0278	61013.81	0.98%

Table A.5 LW algorithm v.s. CPLEX on σ/ μ (S=5 N=3 J=5) (Con't)

Run	σ/μ=20%					σ/μ=25%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.3959	69181.16	0.0400	69381.16	0.29%	0.2006	66375.44	0.0399	66875.44	0.75%
2	0.3476	61800.23	0.0120	62000.23	0.32%	0.2585	83111.1	0.0491	83311.1	0.24%
3	0.1602	54274.71	0.0355	54474.71	0.37%	0.3168	65166.21	0.0221	65732.07	0.87%
4	0.2561	31362.84	0.0157	33787.18	7.73%	0.3036	88562.55	0.0397	88762.55	0.23%
5	0.3870	53254.87	0.0250	55565.94	4.34%	0.2608	52003.16	0.0173	52203.16	0.38%
6	0.2813	94881.67	0.0256	95081.67	0.21%	0.3712	36867.71	0.0239	37067.71	0.54%
7	0.3602	85187.70	0.0337	85387.70	0.23%	0.3314	87510.23	0.0241	92524.84	5.73%
8	0.1574	34082.32	0.0256	35762.07	4.93%	0.2884	47279.06	0.0477	50103.8	5.97%
9	0.3135	64299.89	0.0331	65669.30	2.13%	0.3046	41764.19	0.0345	43073.52	3.14%
10	0.2361	39308.16	0.0114	40938.69	4.15%	0.2786	31185.2	0.0326	33213.34	6.50%
11	0.3619	79870.48	0.0237	80070.48	0.25%	0.2262	57412.3	0.0150	57612.3	0.35%
12	0.2619	76832.62	0.0331	77000.66	0.22%	0.3277	89867.58	0.0189	90067.58	0.22%
13	0.1859	37417.39	0.0284	37617.39	0.53%	0.3811	50358.59	0.0326	53835.04	6.90%
14	0.3856	34376.24	0.0279	34767.49	1.14%	0.3947	78178.1	0.0179	79298.05	1.43%
15	0.3788	81630.41	0.0237	81830.41	0.25%	0.3020	31006.27	0.0330	32988.69	6.39%
16	0.2362	62471.72	0.0138	62671.72	0.32%	0.2616	58027.1	0.0576	58227.1	0.34%
17	0.3852	65387.23	0.0209	65587.23	0.31%	0.3971	63589.94	0.0387	63789.94	0.31%
18	0.3691	86455.78	0.0289	86655.78	0.23%	0.2636	41292.95	0.0108	42328.1	2.51%
19	0.2002	64336.49	0.0327	68379.03	6.28%	0.2024	69704.43	0.0235	69904.43	0.29%

20	0.3054	47339.38	0.0469	50148.47	5.93%	0.1350	59092.55	0.0458	59292.55	0.34%
21	0.3538	43843.12	0.0144	44909.96	2.43%	0.3357	90007.21	0.0216	90207.21	0.22%
22	0.3328	78015.10	0.0440	83307.03	6.78%	0.3876	65456.06	0.0452	69667.12	6.43%
23	0.3593	59323.59	0.0278	61471.45	3.62%	0.3683	44442.55	0.0390	46858.77	5.44%
24	0.1664	50114.53	0.0155	50928.63	1.62%	0.1702	56230.6	0.0241	56878.13	1.15%
25	0.2817	91063.66	0.0119	92062.79	1.10%	0.1607	87273.67	0.0356	88030.55	0.87%
26	0.2787	59544.47	0.0338	61044.50	2.52%	0.3446	40761.28	0.0419	42742.92	4.86%
27	0.3509	72800.86	0.0251	74476.24	2.30%	0.3498	50831.3	0.0522	53404.29	5.06%
28	0.3312	46189.75	0.0136	47581.02	3.01%	0.3261	40576.42	0.0163	42337.54	4.34%
29	0.1690	85532.53	0.0197	88415.16	3.37%	0.1405	70191.03	0.0144	71994.86	2.57%
30	0.2660	59502.63	0.0121	59837.51	0.56%	0.1855	72209.89	0.0118	75743.89	4.89%

Table A.6 LW algorithm v.s. CPLEX on σ/ μ (S=5 N=3 J=5) (Con't)

Run	σ/μ=30%					σ/μ=40%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.2423	83208.56	0.0285	83408.56	0.24%	0.6173	84709.66	0.0295	85642.56	1.10%
2	0.2720	75872.2	0.0173	76072.2	0.26%	0.3517	81124.45	0.0267	81408.9	0.35%
3	0.3962	74839.54	0.0438	75717.62	1.17%	0.2442	69251.75	0.0436	69680.7	0.62%
4	0.2816	92133.84	0.0427	92333.84	0.22%	0.4727	83442	0.0329	84775.95	1.60%
5	0.3711	45875.16	0.0200	45982.05	0.23%	0.5227	64559.55	0.0257	64824.17	0.41%
6	0.4406	46699.1	0.0110	48364.5	3.57%	0.3834	41605.5	0.0212	42640.01	2.49%
7	0.2233	70556	0.0160	72325.13	2.51%	0.2436	49056.6	0.0398	49831.41	1.58%
8	0.2103	75219.64	0.0219	75608.87	0.52%	0.6107	78431.19	0.0390	79502.24	1.37%
9	0.2399	77573.75	0.0341	77684.01	0.14%	0.5682	49436.81	0.0248	49839.5	0.81%
10	0.3681	54783.35	0.0184	57289.99	4.58%	0.3032	56602.09	0.0385	58356.64	3.10%
11	0.3256	38881.69	0.0215	40855.43	5.08%	0.3642	56448.17	0.0420	58383.55	3.43%
12	0.6237	41423.29	0.0206	42983.25	3.77%	0.4984	40946.3	0.0354	41443.01	1.21%
13	0.2839	77199.55	0.0185	79874.31	3.46%	0.4641	73159.54	0.0380	73651.37	0.67%
14	0.2999	49750.17	0.0379	50367.66	1.24%	0.3730	47888.53	0.0346	48372.98	1.01%
15	0.3037	44467.61	0.0255	45749.5	2.88%	0.3412	64137.48	0.0390	64603.59	0.73%
16	0.2526	33823.36	0.0151	34580.41	2.24%	0.3780	43146.05	0.0351	43967.64	1.90%
17	0.3287	50502.1	0.0373	50702.1	0.40%	0.4949	63960.48	0.0276	65933.69	3.09%

18	0.2638	56745.56	0.0423	57684.85	1.66%	0.4280	51831.79	0.0187	52187.95	0.69%
19	0.3695	75740.61	0.0118	78425.43	3.54%	0.3788	83350.06	0.0121	85850.35	3.00%
20	0.3988	39374.89	0.0204	40009.69	1.61%	0.3219	59502.6	0.0206	59852.5	0.59%
21	0.3152	84192.14	0.0194	86412.72	2.64%	0.4812	36135.53	0.0139	36325.47	0.53%
22	0.4950	80774.29	0.0371	82849.22	2.57%	0.4074	71938.34	0.0233	72478.64	0.75%
23	0.4348	63969.14	0.0128	64537.79	0.89%	0.4181	43896.8	0.0343	44786.74	2.03%
24	0.4406	52006.07	0.0175	53054.35	2.02%	0.3219	56833.21	0.0359	58188.26	2.38%
25	0.2945	41470.56	0.0376	41607.56	0.33%	0.3691	54810.7	0.0269	55244.68	0.79%
26	0.2939	45983.21	0.0143	46414.61	0.94%	0.3424	60160.25	0.0190	61445.97	2.14%
27	0.2607	35951.3	0.0231	36272.32	0.89%	0.2655	43331.63	0.0262	44453.36	2.59%
28	0.4487	71449.98	0.0345	73402.26	2.73%	0.4847	63535.99	0.0328	65588.91	3.23%
29	0.5162	49198.64	0.0326	51021.38	3.70%	0.4717	52964.01	0.0338	54193.38	2.32%
30	0.4383	60474.91	0.0347	62227.86	2.90%	0.4383	60474.91	0.0347	61893.46	2.35%

Table A.7 LW algorithm v.s. CPLEX on σ/ μ (S=8 N=5 J=10)

Run	σ/μ=10%					σ/μ=15%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	9.7650	96844.72	0.1023	96999.95	0.16%	9.0731	48242.04	0.1028	50105.72	3.86%
2	8.7963	92429.84	0.1201	93093.65	0.72%	7.2849	74941.7	0.1388	74992.53	0.07%
3	7.0408	88009.71	0.0983	90468.62	2.79%	9.0122	66019.4	0.1162	67801.73	2.70%
4	7.9707	101626.1	0.0921	102466.2	0.83%	8.5117	76700.54	0.1234	80999.31	5.60%
5	10.9751	58902.19	0.1286	58997.93	0.16%	9.4692	38675.97	0.1192	38877.92	0.52%
6	5.9306	61809.02	0.1164	63547.48	2.81%	10.6345	60006.62	0.0924	62310.81	3.84%
7	10.9883	48968.49	0.1353	50396.28	2.92%	10.1467	102610.1	0.0932	102614	0.00%
8	10.3885	86966.76	0.1106	86976.14	0.01%	8.7027	92210.36	0.1127	92808.54	0.65%
9	6.9773	96119.26	0.1027	96273.17	0.16%	7.3211	40132.03	0.1347	40269.74	0.34%
10	6.1298	38992.73	0.1038	39112.47	0.31%	7.0592	71868.28	0.1256	73067.19	1.67%
11	9.5596	81428.61	0.0956	81948.92	0.64%	10.5661	44797.44	0.1376	45060.61	0.59%
12	6.3215	104612.2	0.1261	104921.3	0.30%	6.7698	86867.43	0.1162	89117.45	2.59%
13	7.5464	97575.03	0.1340	98462.38	0.91%	11.1953	82261.24	0.1054	85782.69	4.28%
14	9.2162	91715.57	0.1231	93011.28	1.41%	9.7080	42850.56	0.0905	43445.24	1.39%
15	10.0379	78544.56	0.1201	79317.35	0.98%	9.0514	38994.82	0.1127	40121.43	2.89%

16	11.1239	77739.56	0.1125	79916.7	2.80%	10.8354	64783.85	0.1273	66953.5	3.35%
17	10.1191	69243.89	0.1143	71166.53	2.78%	11.2975	63499.95	0.1377	66617.8	4.91%
18	6.6453	69442.67	0.1339	69859.12	0.60%	6.5552	71965.63	0.1159	72213.16	0.34%
19	8.9968	60618.24	0.1117	63081.36	4.06%	5.8274	90505.85	0.1106	92184.26	1.85%
20	11.0084	81409.6	0.1113	82196.69	0.97%	9.1511	68855.73	0.1290	71307.78	3.56%
21	10.6846	95850.45	0.0936	98189.1	2.44%	10.2380	50318.72	0.1116	52918.06	5.17%
22	10.1061	74353.14	0.1282	77073.11	3.66%	7.8527	95603.77	0.1211	97305.64	1.78%
23	9.0895	67800.59	0.1251	68409.75	0.90%	8.1900	59443.88	0.1304	61528.12	3.51%
24	9.6704	87843.34	0.0964	91143.69	3.76%	7.5958	97707.85	0.0985	100210.6	2.56%
25	6.9492	75238.43	0.1270	75651.85	0.55%	7.4359	75296.67	0.1347	78969.45	4.88%
26	7.2758	55383.49	0.1192	55594.53	0.38%	8.7284	48354.19	0.0930	50565.53	4.57%
27	7.8645	103318.7	0.1075	106350.2	2.93%	6.3476	52168.26	0.1187	52420.36	0.48%
28	7.5188	103000	0.0944	105654.5	2.58%	8.8243	42617.09	0.1201	44413.17	4.21%
29	6.7462	50319.05	0.1222	52345.61	4.03%	9.8213	42178.71	0.1238	42584.6	0.96%
30	7.6990	97296.28	0.1132	99387.54	2.15%	5.8333	53098.88	0.1242	55749.94	4.99%

Table A.8 LW algorithm v.s. CPLEX on σ/ μ (S=8 N=5 J=10) (Con't)

Run	σ/μ=20%					σ/μ=25%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	7.1980	12220.29	0.1324	12538.5	2.60%	5.4011	76135.24	0.1255	76708.8	0.75%
2	7.4761	12674.76	0.1216	13047.17	2.94%	9.9540	66618.93	0.1327	69959.11	5.01%
3	8.5161	11441.66	0.1027	11585.74	1.26%	7.7244	61112.49	0.1307	61858.44	1.22%
4	9.7738	9884.89	0.1232	10089.68	2.07%	9.1228	35954.47	0.1149	37737.79	4.96%
5	7.4282	13972.39	0.1298	14475.24	3.60%	8.2833	57792.72	0.1119	62823.55	8.70%
6	10.5333	10774.91	0.1135	10851.54	0.71%	9.7913	102266.1	0.1322	109581.9	7.15%
7	6.7981	10922.16	0.0826	11107.78	1.70%	9.0897	89196.93	0.1131	92353.84	3.54%
8	6.8199	9767.769	0.1349	10120.1	3.61%	8.7687	40144.07	0.1363	40960.86	2.03%
9	8.8784	11354.25	0.1196	12150.98	7.02%	10.3164	71991.66	0.1278	72091.71	0.14%
10	8.6769	13265.67	0.1198	13785.89	3.92%	7.7509	45281.58	0.1227	48061.09	6.14%
11	7.7307	13350.58	0.0978	13415.94	0.49%	6.2090	85775.14	0.1181	87500.74	2.01%
12	6.4183	10457.43	0.0887	10467.65	0.10%	8.8265	81490.83	0.1004	81591.05	0.12%
13	7.9290	10844.46	0.1161	11074.31	2.12%	6.7080	44194.28	0.0926	44798.86	1.37%

14	5.7874	11154.1	0.1114	11250.4	0.86%	7.2156	40660.14	0.1172	40996.73	0.83%
15	11.2461	13218.85	0.1263	13458.86	1.82%	8.1724	88351.23	0.1100	89489.49	1.29%
16	11.1783	11062.18	0.0989	11683.19	5.61%	10.5777	66799.94	0.1228	66829.6	0.04%
17	6.5740	13709.16	0.0869	14569.52	6.28%	11.0136	76444.93	0.1177	78744.53	3.01%
18	5.8807	9355.62	0.1073	10008.25	6.98%	7.1890	92671.31	0.1148	93685.42	1.09%
19	6.3101	10381.34	0.1153	10678.67	2.86%	10.9081	68213.05	0.1386	73839.61	8.25%
20	9.2900	10283.26	0.1167	10489.7	2.01%	7.5476	55196.11	0.1349	55341.27	0.26%
21	7.9079	11708.94	0.1009	11878.85	1.45%	11.2760	71756.39	0.0965	76156.4	6.13%
22	7.8907	11033.2	0.0969	11097.78	0.59%	10.0723	76889.86	0.1214	81870.5	6.48%
23	6.4820	13924.93	0.1245	14554.7	4.52%	9.0638	59492.53	0.1135	63788.08	7.22%
24	9.3607	10339.79	0.1260	10883.57	5.26%	10.2679	60315.79	0.1283	65278.41	8.23%
25	7.1776	10617.62	0.1279	11317.61	6.59%	7.1263	42491.01	0.1218	45357.01	6.74%
26	10.6331	10551.78	0.1107	10856.54	2.89%	9.7691	82714.72	0.1100	89040.13	7.65%
27	6.8550	11351.85	0.0867	11671.03	2.81%	8.4375	64088.39	0.1315	68280.81	6.54%
28	9.1504	12777.95	0.1129	13478.16	5.48%	6.6777	50964.4	0.1190	54811.72	7.55%
29	7.5063	11430.18	0.0936	11538.93	0.95%	7.4307	52578.88	0.1365	55311.71	5.20%
30	7.4962	9463.022	0.0885	9913.546	4.76%	7.5029	64330.16	0.1222	65039.04	1.10%

Table A.9 LW algorithm v.s. CPLEX on σ/ μ (S=8 N=5 J=10) (Con't)

Run	σ/μ=30%					σ/μ=40%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	8.9593	78634.76	0.1029	78834.76	0.25%	8.6560	74215.24	0.0991	75479.91	1.70%
2	9.6595	84586.61	0.0902	90786.61	7.33%	6.2614	38898.96	0.1303	40100.03	3.09%
3	7.5833	72478.49	0.1195	74599.09	2.93%	7.8360	68715.14	0.1133	72171.49	5.03%
4	6.9254	86351.55	0.1240	86551.55	0.23%	8.7764	68765.22	0.0990	72871.97	5.97%
5	9.8215	74201.07	0.1089	76401.07	2.96%	8.0081	52789.48	0.0991	56760.98	7.52%
6	7.6449	85413.61	0.1266	85613.61	0.23%	9.4252	72110.84	0.1009	75194.05	4.28%
7	10.1300	59448.86	0.1305	62142.56	4.53%	7.2210	70756.91	0.1130	76261.66	7.78%
8	7.4278	65169.57	0.1181	69169.97	6.14%	8.2360	35528.88	0.1330	38318.29	7.85%
9	6.4082	56785.11	0.1092	58792.84	3.54%	9.0569	43847.86	0.1051	45337.39	3.40%
10	10.2925	31968.31	0.1378	33401.12	4.48%	10.1238	51310.15	0.0916	51916.43	1.18%
11	9.1730	65002.77	0.1198	70845.78	8.99%	6.9359	48071.40	0.1122	50183.58	4.39%

12	10.4194	84590.96	0.1255	84790.96	0.24%	10.8380	47351.39	0.1008	50809.58	7.30%
13	8.4442	33545.47	0.1271	34507.35	2.87%	10.6574	87195.98	0.1175	92836.04	6.47%
14	9.7021	69531.63	0.0923	69731.63	0.29%	10.7519	57138.21	0.0989	61441.78	7.53%
15	7.6424	33676.32	0.0995	36110.10	7.23%	6.5338	47234.57	0.1163	49015.56	3.77%
16	6.2993	36239.60	0.1380	38280.22	5.63%	8.3618	63767.82	0.1072	69321.72	8.71%
17	7.7984	77379.46	0.1193	77579.46	0.26%	6.7002	82372.97	0.1324	87820.36	6.61%
18	9.7031	90739.67	0.1015	90939.67	0.22%	5.8512	42622.39	0.1030	45914.34	7.72%
19	7.2688	50759.19	0.1010	51947.88	2.34%	8.0289	44921.65	0.1126	47009.31	4.65%
20	5.7905	84999.71	0.1188	88199.71	3.76%	10.8702	52791.47	0.0935	56275.90	6.60%
21	11.1747	82290.86	0.1183	82987.27	0.85%	7.6854	45745.99	0.1107	47914.82	4.74%
22	10.7082	50585.33	0.0910	50988.77	0.80%	8.3812	78668.67	0.0971	81914.37	4.13%
23	9.3161	51226.39	0.1245	51738.33	1.00%	8.5918	63959.92	0.1258	66221.47	3.54%
24	7.9332	39880.18	0.0950	41453.01	3.94%	9.8756	41801.44	0.0983	43345.37	3.69%
25	10.1766	60578.11	0.1125	65882.78	8.76%	9.0792	54367.30	0.1319	54981.83	1.13%
26	8.4946	74260.20	0.1092	74547.04	0.39%	7.4907	78077.41	0.1205	80937.92	3.66%
27	6.5037	87527.35	0.1350	95343.90	8.93%	9.8244	68976.93	0.1244	71977.44	4.35%
28	10.3259	80172.69	0.1299	85263.35	6.35%	9.5564	74739.98	0.1253	75803.20	1.42%
29	7.9416	40525.68	0.1200	42101.84	3.89%	6.7210	41662.86	0.1333	42421.58	1.82%
30	6.4336	42649.82	0.1370	42869.78	0.52%	6.4336	42649.82	0.1370	43472.04	1.93%

Table A.10 LW algorithm v.s. CPLEX on R (S=5 N=3 J=5)

Run	*r*=100%					*r*=110%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.3447	79938.25	0.0209	83617.64	4.60%	0.2716	106879.2	0.0197	110336.9	3.24%
2	0.3135	72026.74	0.0241	75566.44	4.91%	0.3239	67000.36	0.0260	68964.33	2.93%
3	0.2916	56788.91	0.0357	57784.93	1.75%	0.3264	101416.6	0.0282	105398.7	3.93%
4	0.3053	66869.93	0.0312	69315.85	3.66%	0.2303	80086.56	0.0179	80751.75	0.83%
5	0.3500	79325.36	0.0260	79741.91	0.53%	0.3039	55632.45	0.0229	57116.02	2.67%
6	0.2931	62385.61	0.0345	66107.95	5.97%	0.2872	58344.79	0.0173	60921.46	4.42%
7	0.2433	81310.07	0.0333	82262.51	1.17%	0.3116	86557.3	0.0366	89747.29	3.69%
8	0.2963	67205.72	0.0242	69258.31	3.05%	0.3460	73908.28	0.0213	74626.48	0.97%
9	0.2415	61129.86	0.0317	63106.31	3.23%	0.3064	73516.09	0.0319	74506.51	1.35%

10	0.2370	77267.5	0.0350	79639.19	3.07%	0.3261	87433.21	0.0350	89986.93	2.92%
11	0.3472	76680.22	0.0314	79124.89	3.19%	0.3155	87343.47	0.0334	90940.65	4.12%
12	0.3450	73681.01	0.0161	77929.68	5.77%	0.3230	104904.3	0.0115	105451.9	0.52%
13	0.3141	83914.96	0.0111	84731.38	0.97%	0.2441	100991.5	0.0183	105755.9	4.72%
14	0.3331	71952.04	0.0292	72405.66	0.63%	0.3279	94281.01	0.0345	97300.5	3.20%
15	0.3195	69761.85	0.0360	70168.3	0.58%	0.2984	94707.31	0.0309	98145.75	3.63%
16	0.3059	92033.59	0.0100	93612.51	1.72%	0.2853	76653.52	0.0350	80283.53	4.74%
17	0.3092	78381.17	0.0120	82378.74	5.10%	0.2718	106903.5	0.0104	111257.6	4.07%
18	0.2622	75376.34	0.0190	77257.78	2.50%	0.2897	96324.57	0.0302	97910.61	1.65%
19	0.2329	89947.87	0.0174	93388.85	3.83%	0.2960	55327.41	0.0141	57186.19	3.36%
20	0.3283	97842.86	0.0279	101874.1	4.12%	0.2890	100081.2	0.0173	103298.6	3.21%
21	0.3071	105074.9	0.0164	106445.1	1.30%	0.3061	88725.72	0.0338	92979.79	4.79%
22	0.2979	68365.62	0.0166	70382.66	2.95%	0.3163	58814.06	0.0186	61154.13	3.98%
23	0.2732	86864.43	0.0327	90425.56	4.10%	0.2487	65249.49	0.0167	68004.66	4.22%
24	0.2554	68486.25	0.0158	71805.9	4.85%	0.2468	102898.6	0.0240	105046.4	2.09%
25	0.3035	92406.17	0.0352	96093.26	3.99%	0.2606	65771.29	0.0259	68904.55	4.76%
26	0.3090	69123.4	0.0275	72263.34	4.54%	0.2779	101993.2	0.0255	106680.3	4.60%
27	0.2728	102024	0.0159	104714.6	2.64%	0.2770	81271.34	0.0267	82882.87	1.98%
28	0.2851	61218.53	0.0154	63839.19	4.28%	0.3246	104722.2	0.0234	107693.4	2.84%
29	0.2996	70342.25	0.0315	71580.84	1.76%	0.2453	71908.8	0.0157	74180.74	3.16%
30	0.3118	82680.57	0.0142	85296.22	3.16%	0.2633	93123.19	0.0212	94407.46	1.38%

Table A.11 LW algorithm v.s. CPLEX on R (S=5 N=3 J=5) (Con't)

Run	r=120%					r=130%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.3197	64481.62	0.0212	64481.62	0.00%	0.3152	59655.18	0.0281	62000.05	3.93%
2	0.2563	51114.38	0.0250	51410.23	0.58%	0.3153	52558.77	0.0328	53239.92	1.30%
3	0.2419	69148.89	0.0139	71793.34	3.82%	0.3420	67028.91	0.0247	69782.88	4.11%
4	0.2679	81433.71	0.0320	84230.79	3.43%	0.2832	72293.26	0.0131	75511.21	4.45%
5	0.3344	68290.2	0.0196	71358.67	4.49%	0.2442	64805.71	0.0359	66454.29	2.54%
6	0.2865	74492.28	0.0181	75744.56	1.68%	0.2472	77865.66	0.0352	80812.31	3.78%
7	0.3271	60719.21	0.0336	61842.74	1.85%	0.3177	62033.08	0.0367	64276.97	3.62%

8	0.2572	65889.75	0.0372	68414.64	3.83%	0.2912	62292.49	0.0219	64092.66	2.89%
9	0.2614	75453.51	0.0147	77019.88	2.08%	0.2874	64344.89	0.0297	65543.21	1.86%
10	0.3101	70618.9	0.0348	72995.9	3.37%	0.2832	50640.09	0.0245	51002.91	0.72%
11	0.2370	59690.44	0.0185	59690.44	0.00%	0.3010	72539.15	0.0200	74037.56	2.07%
12	0.2574	58501.31	0.0261	60474.87	3.37%	0.3279	58212.71	0.0212	58212.71	0.00%
13	0.2648	53144.65	0.0318	55177.73	3.83%	0.2573	59305.5	0.0140	59622.29	0.53%
14	0.3168	56519.65	0.0332	57340.8	1.45%	0.2971	53276.96	0.0139	54743.74	2.75%
15	0.3114	53074.66	0.0247	54643.84	2.96%	0.2989	64927.38	0.0243	67350.85	3.73%
16	0.2700	60166.34	0.0182	61645.32	2.46%	0.2509	63834.84	0.0255	66013.69	3.41%
17	0.3335	79345.98	0.0178	82299.52	3.72%	0.3205	56012.21	0.0172	58483.52	4.41%
18	0.3459	72178.95	0.0140	73493.98	1.82%	0.2430	69459.72	0.0232	69459.72	0.00%
19	0.3106	56385.51	0.0352	58849.08	4.37%	0.3229	68582.59	0.0163	70581.37	2.91%
20	0.3463	86290.1	0.0339	88700.85	2.79%	0.2908	53074.04	0.0257	54956.93	3.55%
21	0.3121	65985.38	0.0251	66390.68	0.61%	0.3468	67841.78	0.0370	70480.09	3.89%
22	0.3097	53811.6	0.0260	56077.61	4.21%	0.2561	75117.36	0.0343	78453.33	4.44%
23	0.3386	73661.28	0.0191	75294.63	2.22%	0.2478	75331.98	0.0309	78526.91	4.24%
24	0.3107	69430.11	0.0248	69833.69	0.58%	0.3310	67701.76	0.0219	69613.69	2.82%
25	0.2982	72276.88	0.0217	74817.13	3.51%	0.2922	59555.42	0.0350	61392.75	3.09%
26	0.2701	78781.73	0.0199	79378.16	0.76%	0.2781	55023.37	0.0311	57314.43	4.16%
27	0.3163	75941.21	0.0328	78789.62	3.75%	0.3325	68127.53	0.0335	69899.22	2.60%
28	0.2603	54844.22	0.0177	56783.01	3.54%	0.3371	58257.34	0.0279	58626.7	0.63%
29	0.2594	74656.32	0.0259	74759.73	0.14%	0.3079	69241.43	0.0176	70308.58	1.54%
30	0.2704	79095.02	0.0268	81798.22	3.42%	0.3345	59502.32	0.0340	59960.72	0.77%

Table A.12 LW algorithm v.s. CPLEX on R (S=5 N=3 J=5) (Con't)

Run	*r*=140%					*r*=150%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	0.2733	40577.75	0.0202	40577.75	0.00%	0.2799	55295.57	0.0275	56443.11	2.08%
2	0.2994	41094.25	0.0288	42520.51	3.47%	0.3355	69202.28	0.0367	71731.66	3.66%
3	0.2522	60122.97	0.0142	62054.84	3.21%	0.3375	64471.44	0.0235	65977.67	2.34%
4	0.3250	46090.65	0.0167	47083.08	2.15%	0.2714	52846.16	0.0288	54168.85	2.50%
5	0.3450	50598.92	0.0312	50598.92	0.00%	0.3012	55741.37	0.0354	56809.01	1.92%

6	0.3244	55533.48	0.0335	57345.44	3.26%	0.3453	50732.28	0.0138	51717.69	1.94%
7	0.2312	57301.81	0.0385	58277.2	1.70%	0.3446	64344.46	0.0141	64407.85	0.10%
8	0.3440	59476.97	0.0333	60940.86	2.46%	0.3078	59206.95	0.0229	61305.54	3.54%
9	0.3327	50493.73	0.0349	52405.11	3.79%	0.3329	54532.80	0.0369	55032.91	0.92%
10	0.2394	55659.51	0.0379	57621.78	3.53%	0.3175	57532.05	0.0233	58550.78	1.77%
11	0.2637	54706.5	0.0275	54706.5	0.00%	0.2986	67375.51	0.0276	69211.31	2.72%
12	0.3317	66476.49	0.0341	68667.46	3.30%	0.2450	62844.90	0.0242	62998.43	0.24%
13	0.2397	43638.69	0.0200	44008.95	0.85%	0.3341	69337.82	0.0151	71420.48	3.00%
14	0.3187	67075.55	0.0124	68284.7	1.80%	0.2721	47442.88	0.0128	47871.31	0.90%
15	0.3463	69891.57	0.0352	69891.57	0.00%	0.3025	48088.70	0.0255	48412.83	0.67%
16	0.2810	59874.84	0.0223	61685.61	3.02%	0.2800	57138.74	0.0260	57433.61	0.52%
17	0.2697	64724.41	0.0130	64724.41	0.00%	0.3009	64826.28	0.0227	66506.31	2.59%
18	0.3425	58935.52	0.0356	58935.52	0.00%	0.3229	52866.47	0.0316	54096.03	2.33%
19	0.3232	61093.33	0.0164	62313.99	2.00%	0.2791	60144.23	0.0279	61494.31	2.24%
20	0.2668	45980.74	0.0170	46778.64	1.74%	0.3246	64106.04	0.0171	66163.67	3.21%
21	0.3116	63792.29	0.0369	65659.76	2.93%	0.2704	61408.79	0.0321	62883.41	2.40%
22	0.2684	52838.55	0.0363	53171.17	0.63%	0.2662	61564.62	0.0172	62448.17	1.44%
23	0.3136	62816.41	0.0160	64208.15	2.22%	0.3165	52318.90	0.0247	54079.12	3.36%
24	0.2975	58518.25	0.0187	59875.16	2.32%	0.3157	65195.48	0.0338	66019.44	1.26%
25	0.2331	47066.28	0.0162	47821.48	1.60%	0.3224	61481.10	0.0169	62135.18	1.06%
26	0.3282	46965.65	0.0361	48091.01	2.40%	0.3049	50571.94	0.0293	51131.72	1.11%
27	0.2983	65303.67	0.0336	67371	3.17%	0.3131	57177.55	0.0321	58064.84	1.55%
28	0.3149	53748.59	0.0343	55233.94	2.76%	0.3119	56958.98	0.0222	57852.44	1.57%
29	0.2978	61023.87	0.0282	63079.11	3.37%	0.2973	60089.33	0.0263	60800.79	1.18%
30	0.2972	56100.95	0.0217	56567.83	0.83%	0.2972	56100.95	0.0217	57142.07	1.86%

Table A.13 LW algorithm v.s. CPLEX on R (S=8 N=5 J=10)

Run	*r*=100%					*r*=110%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	5.4223	93202.33	0.0907	94271.92	1.15%	7.6404	71989.34	0.1064	72234.84	0.34%
2	6.3311	64386.39	0.1024	65794.58	2.19%	10.8790	60479.77	0.1317	62045.72	2.59%
3	7.8818	80264.59	0.1298	83936.35	4.57%	10.2878	55895.23	0.1282	58195.66	4.12%

4	6.7382	80295.06	0.1095	81537.53	1.55%	8.6231	43531.21	0.1251	45735.32	5.06%
5	6.3416	82339.35	0.1248	83893.39	1.89%	10.1527	50972.17	0.1168	52471.60	2.94%
6	5.3823	63107.61	0.1003	65399.47	3.63%	6.5199	73041.63	0.1368	73041.63	0.00%
7	7.9897	69312.92	0.1160	72568.57	4.70%	9.3818	50579.30	0.1216	51449.09	1.72%
8	6.5842	86805.00	0.1269	87414.73	0.70%	11.2716	79859.52	0.0965	80985.29	1.41%
9	10.8430	60051.07	0.0970	60375.46	0.54%	8.8891	82086.98	0.1146	82483.99	0.48%
10	7.7207	65804.77	0.1333	69143.61	5.07%	6.7522	86848.30	0.1332	87331.90	0.56%
11	7.9023	78000.65	0.0927	81440.39	4.41%	7.3371	44098.32	0.1208	44098.32	0.00%
12	5.9765	90061.38	0.1317	91444.53	1.54%	9.5766	56559.79	0.1359	58006.47	2.56%
13	10.5310	90581.97	0.0897	91021.71	0.49%	10.4135	61814.77	0.1039	64783.73	4.80%
14	9.2592	76596.30	0.0900	77049.80	0.59%	8.5754	63998.02	0.0968	67440.33	5.38%
15	6.2927	57861.66	0.1135	58060.34	0.34%	10.8716	74581.71	0.1360	75050.21	0.63%
16	5.5287	54711.96	0.1248	56533.50	3.33%	8.1067	44571.62	0.1385	44883.74	0.70%
17	9.3768	84070.55	0.1200	84913.61	1.00%	6.5730	50783.76	0.1052	52963.47	4.29%
18	7.0402	80501.64	0.1214	82256.18	2.18%	7.1678	80531.96	0.1322	84830.47	5.34%
19	7.3406	88522.49	0.0987	88826.24	0.34%	7.7416	75942.78	0.1107	77695.62	2.31%
20	5.2942	67887.62	0.0943	68021.75	0.20%	7.9875	89834.39	0.1020	92012.44	2.42%
21	8.3174	95877.63	0.1168	96512.78	0.66%	8.7228	52829.93	0.1373	55100.23	4.30%
22	8.3190	84539.53	0.1067	87797.32	3.85%	10.8400	89702.18	0.1249	91952.22	2.51%
23	6.3793	77687.74	0.0944	79544.67	2.39%	9.6811	67123.91	0.1195	69882.22	4.11%
24	8.3303	65271.41	0.1032	68557.55	5.03%	10.9366	43655.20	0.1188	44666.82	2.32%
25	7.8808	74529.76	0.1041	75732.32	1.61%	10.9633	43744.83	0.1136	45105.41	3.11%
26	8.4810	80606.31	0.1274	81041.43	0.54%	7.7798	67668.23	0.1357	68317.28	0.96%
27	9.2453	91218.45	0.0995	95453.83	4.64%	7.9454	88904.16	0.1281	92620.30	4.18%
28	7.3968	69408.14	0.1071	71012.63	2.31%	7.2561	85007.20	0.1204	89571.77	5.37%
29	5.3688	86696.44	0.1091	88629.89	2.23%	7.9620	44621.07	0.1304	46954.31	5.23%
30	6.2847	83447.69	0.1249	86856.52	4.08%	7.2607	60923.19	0.1083	64168.66	5.33%

Table A.14 LW algorithm v.s. CPLEX on R (S=8 N=5 J=10) (Con't)

Run	*r*=120%					*r*=130%				
	Cplex		LW		%Gap	Cplex		LW		%Gap
	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	5.5792	47803.44	0.1026	47916.77	0.24%	6.0061	54371.77	0.0991	54971.64	1.10%

2	10.0433	43425.44	0.1208	45434.86	4.63%	10.2039	102352.4	0.1365	106517.2	4.07%
3	10.7245	51871.21	0.0995	53092.76	2.35%	10.0157	65758.19	0.1329	68089.81	3.55%
4	8.4129	54579.15	0.1231	55154.35	1.05%	7.1035	60624.42	0.1015	62077.16	2.40%
5	7.5421	68994.45	0.1075	73070.68	5.91%	8.2233	90070.72	0.0997	91155.09	1.20%
6	7.2714	59446.10	0.1110	62113.77	4.49%	7.9659	81183.54	0.0959	83580.58	2.95%
7	10.8334	47344.05	0.1182	49150.73	3.82%	5.5075	76697.34	0.1219	76993.91	0.39%
8	5.5677	56483.61	0.0903	56483.61	0.00%	11.0019	87322.71	0.1016	90751.51	3.93%
9	10.3908	48157.23	0.1214	49262.72	2.30%	10.0501	97864.02	0.1381	99805.78	1.98%
10	9.0832	44433.58	0.1288	44433.58	0.00%	9.9497	62963.32	0.1049	65373.95	3.83%
11	5.4228	61373.57	0.0896	63223.84	3.01%	10.8536	92262.26	0.0957	93314.75	1.14%
12	10.0205	71263.56	0.1244	73773.53	3.52%	6.6297	60949.78	0.1103	63355.42	3.95%
13	6.8806	73678.70	0.0934	75841.67	2.94%	7.2729	69305.67	0.1271	69495.05	0.27%
14	6.7059	49815.12	0.0943	51486.66	3.36%	7.2793	58626.94	0.0991	58967.98	0.58%
15	10.5372	48776.95	0.1344	49124.31	0.71%	7.8773	48867.1	0.1234	50438.11	3.21%
16	6.8270	61956.30	0.1191	61956.30	0.00%	7.3418	66951.45	0.0914	69108.37	3.22%
17	7.7505	52742.73	0.1337	53155.18	0.78%	7.6291	101996.8	0.1103	106251.8	4.17%
18	9.0073	78297.20	0.0991	81742.82	4.40%	10.4828	100198.1	0.1094	102506.6	2.30%
19	10.8820	78132.57	0.1300	80293.68	2.77%	5.6585	85427.06	0.1330	86666.38	1.45%
20	9.7868	77786.86	0.1209	78627.02	1.08%	8.8266	71791.18	0.0925	72139.76	0.49%
21	8.6284	54885.39	0.1310	57301.67	4.40%	6.7515	77777.72	0.1075	81132.74	4.31%
22	7.4731	48253.96	0.1027	50507.83	4.67%	6.7889	51187.13	0.1114	52165.82	1.91%
23	8.1448	66186.70	0.1269	67347.53	1.75%	6.0911	49326.33	0.1063	49868.19	1.10%
24	5.4426	69571.89	0.0952	72703.39	4.50%	7.1717	90075.03	0.1190	91767.75	1.88%
25	7.2243	57579.64	0.1248	59218.26	2.85%	7.9658	99008.62	0.0920	100207.6	1.21%
26	6.3678	55980.76	0.1038	57010.59	1.84%	8.4940	84112.14	0.1063	84376.75	0.31%
27	6.5678	50335.30	0.0937	52479.27	4.26%	6.9148	69620.89	0.1227	70610.16	1.42%
28	9.4919	44641.86	0.1292	45100.74	1.03%	8.5309	74719.26	0.0988	75119.81	0.54%
29	8.6033	54961.34	0.1293	55377.20	0.76%	8.7792	70362.35	0.1230	71865.63	2.14%
30	9.5907	52345.61	0.1231	54140.51	3.43%	7.5945	93432.34	0.1238	96944.87	3.76%

Table A.15 LW algorithm v.s. CPLEX on R (S=8 N=5 J=10) (Con't)

Run	*r*=140%			*r*=150%		
	Cplex	LW	%Gap	Cplex	LW	%Gap

	CPU	G*	CPU	G^{LW}		CPU	G*	CPU	G^{LW}	
1	11.1322	42703.2	0.0987	42703.2	0.00%	9.7973	65332.09	0.0990	67269.03	2.96%
2	9.5657	46612.38	0.0908	47176.22	1.21%	9.4702	75744.36	0.1347	75769.87	0.03%
3	11.2615	64234.31	0.1218	66232.4	3.11%	9.0770	43927.14	0.0929	44696.62	1.75%
4	7.5291	58681.52	0.1348	60095.38	2.41%	10.7988	71569.28	0.1014	72415.56	1.18%
5	10.9059	63407.06	0.1104	63407.06	0.00%	6.9601	76482.45	0.1219	77294.90	1.06%
6	10.5929	58921.74	0.1226	61177.29	3.83%	8.5941	45140.93	0.1094	45300.95	0.35%
7	7.4667	44008.71	0.1123	44691.88	1.55%	8.0653	73665.67	0.0967	75305.18	2.23%
8	5.6908	62817.75	0.0961	65106.9	3.64%	5.8192	78425.32	0.1095	80788.58	3.01%
9	6.0216	64578.95	0.0925	64578.95	0.00%	7.3246	61458.39	0.1334	63321.85	3.03%
10	9.9973	51333.38	0.0979	52263.19	1.81%	7.8505	69604.38	0.1025	69610.42	0.01%
11	5.7125	79019.71	0.1237	79104.63	0.11%	7.3338	51750.12	0.1327	53067.42	2.55%
12	9.0093	64635.13	0.1274	64851.96	0.34%	8.0814	53932.99	0.1279	54641.52	1.31%
13	6.7093	44894.62	0.1062	44894.62	0.00%	8.1542	46695.37	0.1261	46834.22	0.30%
14	10.0643	41395.19	0.0933	42566.08	2.83%	9.3432	62497.73	0.1270	64508.08	3.22%
15	7.8615	56845.84	0.1275	56845.84	0.00%	6.4941	51510.10	0.1247	51575.97	0.13%
16	5.8667	62533.67	0.1266	62533.67	0.00%	6.8607	62117.29	0.1112	64322.40	3.55%
17	10.5356	56077.16	0.1029	58010.27	3.45%	9.1847	44924.74	0.1068	45400.47	1.06%
18	10.4475	47840.45	0.1235	49555.87	3.59%	7.1596	66736.73	0.1323	68040.77	1.95%
19	10.1706	54543.92	0.1047	54543.92	0.00%	8.5197	73043.42	0.1267	74030.62	1.35%
20	6.8930	58623.42	0.1340	60704.32	3.55%	8.7745	51240.30	0.1113	52409.93	2.28%
21	6.9267	42443.99	0.1343	42551.16	0.25%	7.8774	69182.33	0.1260	71332.53	3.11%
22	9.7120	44003.53	0.1068	45048.49	2.37%	6.9536	62853.65	0.1293	63496.52	1.02%
23	8.2690	52875.29	0.1346	53342.44	0.88%	6.7307	61150.59	0.1252	62783.25	2.67%
24	7.1051	72603.9	0.1041	72894.99	0.40%	10.1853	61314.34	0.1154	63273.00	3.19%
25	10.4622	66859.12	0.1335	67672.95	1.22%	10.2966	66263.57	0.1084	67225.24	1.45%
26	8.0846	61682.91	0.1026	63745.64	3.34%	8.1839	69033.65	0.1115	70396.76	1.97%
27	8.4670	55913.37	0.1045	56956.03	1.86%	8.2142	56930.67	0.1089	57477.23	0.96%
28	9.0375	68344.86	0.1125	69902.67	2.28%	6.3464	66088.17	0.1110	66901.30	1.23%
29	6.3382	58799.21	0.0998	60569.3	3.01%	7.3429	69576.60	0.1053	71660.89	3.00%
30	10.1803	74147.57	0.1070	74790.84	0.87%	10.1803	74147.57	0.1070	74978.00	1.12%

Appendix B CPLEX code for Algorithm LW

InputDataReader.java

```
// --------------------------------------------------------------------------
// File: examples/src/InputDataReader.java
// Version 10.1
// --------------------------------------------------------------------------
//  Copyright (C) 2001-2006 by ILOG.
//  All Rights Reserved.
//  Permission is expressly granted to use this example in the
//  course of developing applications that use ILOG products.
// --------------------------------------------------------------------------
//
// This is a helper class used by several examples to read input data files
// containing arrays in the format [x1, x2, ..., x3].  Up to two-dimensional
// arrays are supported.
//
package com.dragon.scm;

import java.io.*;
import java.util.*;

public class InputDataReader {
	public static class InputDataReaderException extends Exception {
		InputDataReaderException(String file) {
			super("'" + file + "' contains bad data format");
		}
	}

	StreamTokenizer _tokenizer;
	Reader _reader;
	String _fileName;

	public InputDataReader(String fileName) throws IOException {
		_reader = new FileReader(fileName);
		_fileName = fileName;

		_tokenizer = new StreamTokenizer(_reader);

		// State the '"', '\' as white spaces.
		_tokenizer.whitespaceChars('"', '"');
		_tokenizer.whitespaceChars('\\', '\\');

		// State the '[', ']' as normal characters.
		_tokenizer.ordinaryChar('[');
		_tokenizer.ordinaryChar(']');
		_tokenizer.ordinaryChar(',');
	}

	protected void finalize() throws Throwable {
```

```
		_reader.close();
	}

	double readDouble() throws InputDataReaderException, IOException {
		int ntType = _tokenizer.nextToken();

		if (ntType != StreamTokenizer.TT_NUMBER)
			throw new InputDataReaderException(_fileName);

		return _tokenizer.nval;
	}

	int readInt() throws InputDataReaderException, IOException {
		int ntType = _tokenizer.nextToken();

		if (ntType != StreamTokenizer.TT_NUMBER)
			throw new InputDataReaderException(_fileName);

		return (new Double(_tokenizer.nval)).intValue();
	}

	String readString() throws InputDataReaderException, IOException {
		int ntType = _tokenizer.nextToken();

		if (ntType != StreamTokenizer.TT_WORD)
			throw new InputDataReaderException(_fileName);

		return _tokenizer.sval;
	}

	double[] readDoubleArray() throws InputDataReaderException, IOException {
		int ntType = _tokenizer.nextToken(); // Read the '['

		if (ntType != '[')
			throw new InputDataReaderException(_fileName);

		Vector values = new Vector();
		ntType = _tokenizer.nextToken();
		while (ntType == StreamTokenizer.TT_NUMBER) {
			values.add(new Double(_tokenizer.nval));
			ntType = _tokenizer.nextToken();

			if (ntType == ',') {
				ntType = _tokenizer.nextToken();
			} else if (ntType != ']') {
				throw new InputDataReaderException(_fileName);
			}
		}

		if (ntType != ']')
			throw new InputDataReaderException(_fileName);
```

```
		// Fill the array.
		double[] res = new double[values.size()];
		for (int i = 0; i < values.size(); i++) {
			res[i] = ((Double) values.elementAt(i)).doubleValue();
		}

		return res;
	}

	double[][] readDoubleArrayArray() throws InputDataReaderException,
			IOException {
		int ntType = _tokenizer.nextToken(); // Read the '['

		if (ntType != '[')
			throw new InputDataReaderException(_fileName);

		Vector values = new Vector();
		ntType = _tokenizer.nextToken();

		while (ntType == '[') {
			_tokenizer.pushBack();

			values.add(readDoubleArray());

			ntType = _tokenizer.nextToken();
			if (ntType == ',') {
				ntType = _tokenizer.nextToken();
			} else if (ntType != ']') {
				throw new InputDataReaderException(_fileName);
			}
		}

		if (ntType != ']')
			throw new InputDataReaderException(_fileName);

		// Fill the array.
		double[][] res = new double[values.size()][];
		for (int i = 0; i < values.size(); i++)
			res[i] = (double[]) values.elementAt(i);

		return res;
	}

	int[] readIntArray() throws InputDataReaderException, IOException {
		int ntType = _tokenizer.nextToken(); // Read the '['

		if (ntType != '[')
			throw new InputDataReaderException(_fileName);

		Vector values = new Vector();
		ntType = _tokenizer.nextToken();
		while (ntType == StreamTokenizer.TT_NUMBER) {
```

```
				values.add(new Double(_tokenizer.nval));
				ntType = _tokenizer.nextToken();

				if (ntType == ',') {
					ntType = _tokenizer.nextToken();
				} else if (ntType != ']') {
					throw new InputDataReaderException(_fileName);
				}
			}

			if (ntType != ']')
				throw new InputDataReaderException(_fileName);

			// Fill the array.
			int[] res = new int[values.size()];
			for (int i = 0; i < values.size(); i++)
				res[i] = ((Double) values.elementAt(i)).intValue();

			return res;
		}

		int[][] readIntArrayArray() throws InputDataReaderException, IOException {
			int ntType = _tokenizer.nextToken(); // Read the '['

			if (ntType != '[')
				throw new InputDataReaderException(_fileName);

			Vector values = new Vector();
			ntType = _tokenizer.nextToken();

			while (ntType == '[') {
				_tokenizer.pushBack();

				values.add(readIntArray());

				ntType = _tokenizer.nextToken();
				if (ntType == ',') {
					ntType = _tokenizer.nextToken();
				} else if (ntType != ']') {
					throw new InputDataReaderException(_fileName);
				}
			}

			if (ntType != ']')
				throw new InputDataReaderException(_fileName);

			// Fill the array.
			int[][] res = new int[values.size()][];
			for (int i = 0; i < values.size(); i++)
				res[i] = (int[]) values.elementAt(i);

			return res;
```

```
}

String[] readStringArray() throws InputDataReaderException, IOException {
	int ntType = _tokenizer.nextToken(); // Read the '['

	if (ntType != '[')
		throw new InputDataReaderException(_fileName);

	Vector values = new Vector();
	ntType = _tokenizer.nextToken();
	while (ntType == StreamTokenizer.TT_WORD) {
		values.add(_tokenizer.sval);
		ntType = _tokenizer.nextToken();

		if (ntType == ',') {
			ntType = _tokenizer.nextToken();
		} else if (ntType != ']') {
			throw new InputDataReaderException(_fileName);
		}
	}

	if (ntType != ']')
		throw new InputDataReaderException(_fileName);

	// Fill the array.
	String[] res = new String[values.size()];
	for (int i = 0; i < values.size(); i++)
		res[i] = (String) values.elementAt(i);

	return res;
}

String[][] readStringArrayArray() throws InputDataReaderException,
		IOException {
	int ntType = _tokenizer.nextToken(); // Read the '['

	if (ntType != '[')
		throw new InputDataReaderException(_fileName);

	Vector values = new Vector();
	ntType = _tokenizer.nextToken();

	while (ntType == '[') {
		_tokenizer.pushBack();

		values.add(readStringArray());

		ntType = _tokenizer.nextToken();
		if (ntType == ',') {
			ntType = _tokenizer.nextToken();
		} else if (ntType != ']') {
			throw new InputDataReaderException(_fileName);
```

```
				}
			}

			if (ntType != ']')
				throw new InputDataReaderException(_fileName);

			// Fill the array.
			String[][] res = new String[values.size()][];
			for (int i = 0; i < values.size(); i++)
				res[i] = (String[]) values.elementAt(i);

			return res;
		}
}
```

AlgorithmLW.java

```
/**
 * Multi-modal transportation problem implemented by Cplex-java
 */
package com.dragon.scm;

import ilog.concert.IloException;
import ilog.concert.IloNumExpr;
import ilog.concert.IloNumVar;
import ilog.cplex.IloCplex;

/**
 * @author Gang Wang Date: 4/11/2012
 */
public class Algorithm {

		static double optimalValue;
		static double[][] solution;
		static double[][] assigDCCus;
		static double[][] assign;
		static double[][] supply;

		static class Data {

			int nSuppliers;// the number of suppliers
			int nDCs;// the number of distribution centers
			int nCustomers;// the number of customers asking for demand

			double[] capacitySupplier;// the capacity of each supplier
			double[] capacityDC;// the capacity of each distribution center
			double[] demandMeans;// the mean value of random demand
			double[] penaltyCost;// penalty cost incurred by the unsatisfied demand
			double[] deadlines;// the deadlines of receiving demands for customers
			double[] processingTimes;// the time of processing products at DC
```

```
double[][] shipCostFromSupDC;// the shipping cost from supplier to
// distribution center
double[][] shipCostFromDCCustomer;// shipping cost from distribution
// center to customer
double[][] fixedCostFromDCCustomer;// fixed shipping cost from DC to
// customer

double[][] shippingTimeFromSupDC;// shipping time from supplier to DC
double[][] shippingTimeFromDCCus;// shipping time from DC to customer

double[] fixedCostDCs;

/*****************************************************************
 * read data from file and construct the model parameters *
 * *****************************************************************/

Data(String filename) throws IloException, java.io.IOException,
        InputDataReader.InputDataReaderException {

    /*****************************************************************
     * Construct the object of InputDataReader. Here filename is the
     * file name and not path
     *****************************************************************/
    InputDataReader reader = new InputDataReader(filename);

    capacitySupplier = reader.readDoubleArray();// value the capacities
    // of suppliers
    capacityDC = reader.readDoubleArray();// value the capacities of DCs
    demandMeans = reader.readDoubleArray();// value the mean values of
    // demands
    penaltyCost = reader.readDoubleArray();// value the penalty costs of
    // customers
    deadlines = reader.readDoubleArray();// value the deadlines of
    // customers
    processingTimes = reader.readDoubleArray();// value the processing
    // time of DCs

    // value the shipping costs from suppliers to DCs
    shipCostFromSupDC = reader.readDoubleArrayArray();
    // value the shipping cost from DCs to customers
    shipCostFromDCCustomer = reader.readDoubleArrayArray();
    // value the fixed costs from DCs to Customers
    fixedCostFromDCCustomer = reader.readDoubleArrayArray();
    // value the shipping time from Suppliers to DCs
    shippingTimeFromSupDC = reader.readDoubleArrayArray();
    // value the shipping time from DCs to Customers
    shippingTimeFromDCCus = reader.readDoubleArrayArray();

    fixedCostDCs = reader.readDoubleArray();

    nSuppliers = capacitySupplier.length;
    nDCs = capacityDC.length;
```

```
                nCustomers = demandMeans.length;
        }
}

static void knapsack(Data data, double[] shipCostToCus, int index) {
        double max1 = 0, max2 = 0;
        try {

                IloCplex cplex = new IloCplex();
                IloNumVar[] xCustomer = cplex.boolVarArray(data.nCustomers);
                IloNumVar[] assignment = xCustomer;

                /******** Define the arrays of IloNumExpr type *********/
                IloNumExpr[] exprCustomer1 = new IloNumExpr[data.nCustomers];
                IloNumExpr[] exprCustomer2 = new IloNumExpr[data.nCustomers];
                IloNumExpr[] summation = new IloNumExpr[2];
                /****************************************************/

                for (int i = 0; i < data.nCustomers - 1; i++) {
                        if (Math.max(shipCostToCus[i], shipCostToCus[i + 1]) >= max1)
                                max1 = Math.max(shipCostToCus[i], shipCostToCus[i + 1]);
                        if (Math.max(data.penaltyCost[i], data.penaltyCost[i + 1]) >= max2)
                                max2 = Math.max(data.penaltyCost[i],
                                                data.penaltyCost[i + 1]);
                }

                /** Add the second term in the objective function *******/

                for (int j = 0; j < data.nCustomers; j++) {
                        exprCustomer1[j] = cplex.prod(max1 - shipCostToCus[j],
                                        assignment[j]);
                }

                summation[0] = cplex.sum(exprCustomer1);
                /*****************************************************/

                /** Add the third term in the objective function *******/
                double[] penaltyCost = new double[data.nCustomers];
                double totalPenaltyCost = 0;

                for (int j = 0; j < data.nCustomers; j++) {
                        penaltyCost[j] = (max2 - data.penaltyCost[j])
                                        * data.demandMeans[j] * (-1);
                        totalPenaltyCost += penaltyCost[j] * (-1);
                }

                for (int j = 0; j < data.nCustomers; j++) {
                        exprCustomer2[j] = cplex.prod(penaltyCost[j], assignment[j]);
                }

                summation[1] = cplex.sum(exprCustomer2);
                /*****************************************************/
```

```
                // create the objective function
                cplex
                        .addMaximize(cplex.sum(totalPenaltyCost, cplex
                                .sum(summation)));
                // constraint 5
                IloNumExpr[] tempConstraint5 = new IloNumExpr[data.nCustomers];
                for (int j = 0; j < data.nCustomers; j++) {
                    tempConstraint5[j] = cplex.prod(data.demandMeans[j],
                            assignment[j]);
                }
                cplex.addLe(cplex.sum(tempConstraint5), data.capacityDC[index]
                        / (1 + 0.98));
                if (cplex.solve()) {
                    for (int j = 0; j < data.nDCs; j++) {
                        assigDCCus[j] = cplex.getValues(assignment);
                    }
                }
                cplex.end();

            } catch (IloException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        static void scheduling(Data data) {
            try {
                IloCplex cplex = new IloCplex();

                int nSuppliers = data.nSuppliers;
                int nDCs = data.nDCs;
                int nCustomers = data.nCustomers;

                IloNumVar[][] Assign = new IloNumVar[nSuppliers][nDCs];
                IloNumVar[][] Supply = new IloNumVar[nSuppliers][nDCs];

                for (int s = 0; s < nSuppliers; s++) {
                    Assign[s] = cplex.boolVarArray(nDCs);
                    Supply[s] = cplex.numVarArray(nDCs, 0, Double.MAX_VALUE);
                }

                /******** Define the arrays of IloNumExpr type *********/
                IloNumExpr[] exprSupplier = new IloNumExpr[nSuppliers];
                /****************************************************/

                /** Add the first term in the objective function *******/
                for (int i = 0; i < nSuppliers; i++) {
                    exprSupplier[i] = cplex.scalProd(data.shipCostFromSupDC[i],
                            Supply[i]);
                }
                /*****************************************************/
```

```
// create the objective function
cplex.addMinimize(cplex.sum(exprSupplier));
assigDCCus[2][1]=0;
// constraint 1
for (int i = 0; i < nSuppliers; i++) {
        cplex.addRange(0, cplex.sum(Supply[i]),
                        data.capacitySupplier[i]);
}
for (int i = 0; i < nSuppliers; i++) {
        for (int j = 0; j < nDCs; j++) {
                cplex.addLe(Supply[i][j], cplex.prod(Assign[i][j],
                                data.capacitySupplier[i]));
                cplex.addLe(cplex.prod(Assign[i][j], 1.0), Supply[i][j]);
        }
}

// Constraint 2: Flow balance constraints for DCs

for (int n = 0; n < nDCs; n++) {
        double sum = 0;
        IloNumExpr[] epxrShip = new IloNumExpr[nSuppliers];
        for (int s = 0; s < nSuppliers; s++) {
                epxrShip[s] = cplex.prod(1.0, Supply[s][n]);
        }
        for (int j = 0; j < nCustomers; j++) {
                sum += data.demandMeans[j] * assigDCCus[n][j];
        }
        cplex.addEq(cplex.sum(epxrShip), sum);
}

// Constraint 4: Time window constraint for customers
IloNumExpr[] epxrSupply = new IloNumExpr[nSuppliers];
for (int n = 0; n < nDCs; n++) {
        for (int s = 0; s < nSuppliers; s++) {
                epxrSupply[s] = cplex.prod(
                                data.shippingTimeFromSupDC[s][n],
Assign[s][n]);
        }
        for (int j = 0; j < nCustomers; j++) {
                if (assigDCCus[n][j] == 1.0)
                        cplex
                                        .addLe(

    cplex.sum(cplex.max(epxrSupply),

    data.processingTimes[n]),
                                                        (data.deadlines[j]
- data.shippingTimeFromDCCus[n][j])

    * assigDCCus[n][j]);
        }
```

```
            }

            if (cplex.solve()) {
                optimalValue += cplex.getObjValue();
                for (int i = 0; i < nSuppliers; i++) {
                    // assign[i] = cplex.getValues(Assign[i]);
                    supply[i] = cplex.getValues(Supply[i]);
                }
            }

            cplex.end();
        } catch (IloException ex) {
            System.out.println("Concert Error: " + ex);
        }
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
            String filename = "gap.dat";
            Data data = new Data(filename);

            int nSuppliers = data.nSuppliers;
            int nDCs = data.nDCs;
            int nCustomers = data.nCustomers;

            double[][] shipCostFromDCCustomer = new double[nDCs][nCustomers];
            double[][] fixedCostFromDCCustomer = new double[nDCs][nCustomers];
            double[] demandMeans = new double[nCustomers];
            double[][] shipCostToCus = new double[nDCs][nCustomers];

            shipCostFromDCCustomer = data.shipCostFromDCCustomer;
            fixedCostFromDCCustomer = data.fixedCostFromDCCustomer;
            demandMeans = data.demandMeans;

            // optimal solution matrix
            assigDCCus = new double[nDCs][nCustomers];

            // show the assignment of customers to DCs and initialization
            int[] indicator = new int[data.nCustomers];
            for (int j = 0; j < indicator.length; j++) {
                indicator[j] = -1;
            }

            // build initial cost matrix
            for (int n = 0; n < nDCs; n++) {
                for (int j = 0; j < nCustomers; j++) {
                    shipCostToCus[n][j] = shipCostFromDCCustomer[n][j]
```

```
                                                  * demandMeans[j] +
fixedCostFromDCCustomer[n][j];
                    }
               }

               // create cost matrix in recursion j
               // IloNumVar[][] assignment = new IloNumVar[nDCs][nCustomers];
               double[][] shipCostToCusTmp = new double[nDCs][nCustomers];
               shipCostToCusTmp = shipCostToCus;

               for (int n = 0; n < nDCs; n++) {
                    for (int j = 0; j < nCustomers; j++) {
                         if (indicator[j] != -1)
                              shipCostToCus[n][j] = shipCostToCus[n][j]
                                        - shipCostToCus[n][indicator[j]];
                    }
                    knapsack(data, shipCostToCus[n], n);
                    for (int j = 0; j < nCustomers; j++) {
                         if ((new Double(1.0)).equals(assigDCCus[n][j]))
                              indicator[j] = n;
                    }
               }

               for (int j = 0; j < nCustomers; j++) {
                    System.out.println(indicator[j]);
               }

               assigDCCus = new double[nDCs][nCustomers];
               for (int j = 0; j < nCustomers; j++) {
                    if (indicator[j] != -1)
                         assigDCCus[indicator[j]][j] = 1.0;
               }

               for (int n = 0; n < nDCs; n++) {
                    for (int j = 0; j < nCustomers; j++) {
                         optimalValue += shipCostToCusTmp[n][j] *
assigDCCus[n][j];
                    }
               }

               for (int j = 0; j < nCustomers; j++) {
                    double sum = 0;
                    for (int n = 0; n < nDCs; n++) {
                         sum += assigDCCus[n][j];
                    }
                    optimalValue += data.penaltyCost[j] * data.demandMeans[j]
                              * (1 - sum);
               }

               // schedule the transportation between suppliers and DCs

               assign = new double[nSuppliers][nDCs];
```

```
                    supply = new double[nSuppliers][nDCs];

                    scheduling(data);

                    System.out
                              .println("--------------------------------------------");

                    System.out.println("Solution value  = " + optimalValue);

                    System.out
                              .println("--------------------------------------------");

                    System.out
                              .println("--------Assignment of DCs to Customers---------");
                    for (int n = 0; n < nDCs; n++) {
                         for (int j = 0; j < data.nCustomers; ++j)
                              System.out.println("DC-Customer : (" + n + "," + j
                                        + ") assignment = " + assigDCCus[n][j]);
                    }

                    System.out
                              .println("--------Assignment of Suppliers to DCs---------");

                    for (int i = 0; i < nSuppliers; i++) {
                         for (int j = 0; j < nDCs; ++j) {
                              System.out.println("Supplier-DC : (" + i + "," + j
                                        + ") Shipping Quantity = " + assign[i][j]);
                         }
                    }

                    System.out.println("--------Shipment of Suppliers to DCs---------");

                    for (int i = 0; i < nSuppliers; i++) {
                         for (int j = 0; j < nDCs; ++j) {
                              System.out.println("Supplier-DC : (" + i + "," + j
                                        + ") Value = " + supply[i][j]);
                         }
                    }

               } catch (IloException ex) {
                    System.out.println("Concert Error 1: " + ex);
               } catch (InputDataReader.InputDataReaderException ex) {
                    System.out.println("Data Error: " + ex);
               } catch (java.io.IOException ex) {
                    System.out.println("IO Error: " + ex);
               }

          }
     }
```

Curriculum Vita

Gang Wang

1980	Born in June 29, Liaoning Province, China
1999	B.S. , Dalian University of Technology, China
2003-2005	M.S., Dalian University of Technology, China
2005-2008	Ph.D. Student, Dalian University of Technology, China
2008	Ph.D. in Operations Research, Dalian University of Technology, China
2007-2012	Ph.D. Student in Supply Chain Management, Rutgers University
2007-2011	Teaching Assistantship, Rutgers University
2011-2012	Dissertation Fellowship, Rutgers University
2012	Ph.D. in Supply Chain Management, Rutgers University