

**MULTI-ITEM MULTI-ATTRIBUTE OUTSOURCING
IN MANUFACTURING SUPPLY NETWORKS**

by

Seerong Prichanont

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Industrial Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2002

To my parents

ACKNOWLEDGMENTS

My experiences in the University of Wisconsin-Madison have been invaluable. I gratefully acknowledge the Royal Thai Government for its financial support for my research.

I have been very fortunate to work with my advisors, Professor Dharmaraj Veeramani and Professor Andrew J. Miller. I would like to express my deepest gratitude to them for their helpful guidance and encouragement. Professor Veeramani's inclination to perfectionism has helped me grow to be a better person. Professor Miller's insightful comments and suggestions have helped me do more solid research.

I am obliged to my committee members, Professor Robert R. Meyer, Professor James A. Rappold, and Professor Harold J. Steudel, for their interest in my work. My special appreciation is extended to Professor Meyer who has generously allocated his time giving precious advice to me.

I am so grateful to have a chance to know my lab mates and colleagues in Madison. A special thanks goes to Krung Sinapiromsaran, Ananth Krishnamurthy, and Debasis Mishra for their sincere friendship and useful academic counsel.

My research would not have been possible without the understanding and patience of my wife, Karndee, who gave birth to our adorable two-month-old daughter, Podi. Karndee deserves more than a few lines of acknowledgment.

Foremost, I thank my parents, who perpetually give me immeasurable love and support. Without them, I certainly would not have been able to finish this last sentence of my dissertation.

ABSTRACT

We consider a multi-attribute sourcing economy consisting of an Original Equipment Manufacturer (OEM) and a set of suppliers. Price and delivery date are two factors that influence the OEM's decision on supplier selection. The OEM demands a set of indivisible items, each has to be delivered on an allowable delivery date. We define a *tuple* as a bundle of items and the associated delivery dates of each item in the bundle.

From the economy's perspective, we want to have a mechanism that efficiently assigns tuples to suppliers in equilibrium. Assuming that the OEM privately knows his valuations on all possible tuples and each supplier privately knows his costs on all possible tuples, we first show that a *competitive equilibrium* always exists, and that the competitive equilibrium prices are non-linear and non-anonymous. Second, we develop a primal-dual-based auction mechanism and show that it terminates in an approximate efficient assignment at a competitive equilibrium. A bound on the inefficiency is proved.

The auction mechanism above requires each supplier to determine all tuples' costs. To achieve that, we represent the supplier's cost models as capacitated lot-sizing problems and propose heuristic algorithms for determining the optimal production plan that minimizes the total cost, consisting of setup, backorder, and penalty costs. We consider two capacitated lot-sizing problems, called CMB and CMBI. While the first problem does not have the integral production quantity restriction, the second one does. The algorithms for CMB prescribe different strategies for the selection of valid inequalities to be added to the problem. For CMBI, an MIP-based algorithm is proposed. We conduct computational experiments that show that, in terms of solving time, our heuristics outperform a commercial solver.

DISCARD THIS PAGE

TABLE OF CONTENTS

	Page
ABSTRACT	iii
LIST OF TABLES	vii
LIST OF FIGURES	viii
I Introduction	1
1 Outsourcing Decisions	2
1.1 Chapter Overview	2
1.2 Basics of Outsourcing	2
1.3 Electronic Outsourcing	3
1.4 Multi-Attribute Outsourcing	4
1.5 Price-Delivery Date Outsourcing	4
1.6 Organization of Document	5
2 Research Summary	6
2.1 Chapter Overview	6
2.2 A Brief Model Description	6
2.3 Research Objectives	7
2.4 Summary of Research Contributions	7
2.4.1 Auction Design	7
2.4.2 Cost Determination Models and Algorithms	8
II Auction Design	9
3 Background on Auctions	10
3.1 Chapter Overview	10

	Page
3.2 Mechanism Design and Auctions	10
3.3 Single-Item Auctions	12
3.4 Multi-Item Auctions	13
3.4.1 Vickrey-Clarke-Groves Mechanism	14
3.4.2 Iterative Mechanisms	17
3.5 Reverse Auctions	19
4 Proposed Auction Algorithm	20
4.1 Chapter Overview	20
4.2 The Model	20
4.3 Linear Programming Formulations	23
4.4 Prices	28
4.5 Existence of Competitive Equilibrium	30
4.6 Iterative Auction Algorithm	32
4.6.1 Proposed Algorithm	32
4.6.2 LP Interpretations of the Algorithm's Outputs	38
4.6.3 Properties of the Algorithm	40
4.7 Practical Significance of the Proposed Auction Algorithm	45
 III Supplier's Cost Determination Models and Algorithms 46	
5 Literature Survey on Lot-Sizing Problems	47
5.1 Chapter Overview	47
5.2 Dynamic Lot-Sizing Problems	48
5.3 Single-Level Single-Item Models	51
5.4 Single-Level Multi-Item Models	55
5.5 Multi-Level Models	63
5.6 Forecast and Planning Horizons	66
5.7 Summary	68
6 Solving the Capacitated Multi-Item Lot-Sizing Problem with Backorder (CMB) Using Valid Inequalities	70
6.1 Chapter Overview	70
6.2 The Model	71
6.3 Valid Inequalities	73
6.4 Proposed Valid Inequality Selection Strategies	75

	Page
6.4.1 All Optimal Point Violated Cuts Strategy (All-V)	77
6.4.2 Most Violated Strategy (Most-V)	79
6.4.3 Maximum Number of Extreme Points Cut Off Strategy (Max-XP) . .	81
6.5 Implementations and Computational Results	85
6.5.1 Solving Strategies Considered	85
6.5.2 Test Problems	87
6.5.3 Computational Results	88
7 Solving the Capacitated Multi-Item Lot-Sizing Problem with Backorder and Integral Unit Production (CMBI) Using an MIP-Based Heuristic .	97
7.1 Chapter Overview	97
7.2 Mathematical Formulation	98
7.3 Existing Heuristics for Complex Problems	99
7.3.1 Local Search Heuristics	100
7.3.2 MIP-Based Heuristics	101
7.4 Cut-Relax-Fix-and-Free Heuristic	102
7.5 Implementations and Computational Results	106
7.5.1 Solving Strategies Considered	106
7.5.2 Test Problems	107
7.5.3 Results	108
IV Future Research	114
8 Future Research	115
8.1 Chapter Overview	115
8.2 Outsourcing Mechanisms	115
8.3 Improvements in Cost Determination Algorithms	116
LIST OF REFERENCES	118

DISCARD THIS PAGE

LIST OF TABLES

Table	Page
4.1 Progress of the algorithm in Example 2	37
6.1 Notation necessary to describe Max-XP solution method	82
6.2 Parameter values used in the experiment	88
6.3 Computing time (seconds) excluding LSB valid inequalities selection time	91
6.4 Normalized computing time excluding LSB inequalities selection time	92
6.5 Computing time (seconds) including LSB inequalities selection time	93
6.6 Normalized computing time including LSB inequalities selection time	94
6.7 Optimality gap achieved (percent)	95
6.8 Best objective found	96

DISCARD THIS PAGE

LIST OF FIGURES

Figure	Page
4.1 OUTSOURCING algorithm	34
6.1 The LSB inequality selection in All-V, Most-V, and Max-XP	78
7.1 Hypothetical z_i change over time	105
7.2 The change of terminal optimality gap over time for Problems 1 and 2	109
7.3 The change of terminal optimality gap over time for Problems 3 and 4	110
7.4 The change of terminal optimality gap over time for Problems 5 and 6	111
7.5 The change of terminal optimality gap over time for Problems 7 and 8	112
7.6 The change of terminal optimality gap over time for Problems 9 and 10	113

Part I

Introduction

Chapter 1

Outsourcing Decisions

1.1 Chapter Overview

This chapter briefly discusses a big picture of the outsourcing process. Section 1.2 defines outsourcing and discusses two outsourcing methods. Section 1.3 explains how computing technology makes outsourcing process more effective. In Section 1.4, the benefits of multi-attribute outsourcing are discussed. Section 1.5 describes the motivation underlying our choice of price and delivery date as the two decision factors in supplier selection. In Section 1.6, we present the structure of this document.

1.2 Basics of Outsourcing

Outsourcing is a purchasing process designed to maximize delivered value. Companies may outsource products or services through a variety of trading mechanisms, such as negotiations, auctions, or any mix of these. Unfortunately, there is no trading mechanism that is suitable in all situations. Decisions have to be made as to which mechanism is to be used under which situation.

Negotiations are appropriate in a situation where a sourcing company wants to outsource the products or services that are critical to its business operations and/or there are only small number of suppliers who have satisfactory qualifications. Through negotiations, the sourcing company has a chance to get to know its suppliers better, and hence, a trusted business relationship can be established. However, a company considering negotiation as an

outsourcing mechanism should be aware that negotiations usually require longer sourcing cycle times than other mechanisms.

Auctions are appropriate in a situation where a sourcing company wants to outsource the products or services that are not so critical to its business operations. Auctions work well in competitive environments, i.e., environments with many suppliers. A sourcing company that has contracts with some suppliers may hold an auction from time to time to make sure that the negotiated contracts are still attractive.

One characteristic of traditional sourcing processes is that, very often, price is the *only* decision factor sourcing companies use to evaluate suppliers. In some cases, price is the main decision factor, upon which a sourcing company and a supplier have to agree first. Other decision factors will then be negotiated. By putting much emphasis on price, other important decision factors, such as quality of the products, terms of warranty, delivery date, and so on, are often compromised. In Section 1.4 we discuss merits of multi-attribute outsourcing.

1.3 Electronic Outsourcing

Recent development in computing technology allows negotiations and auctions to be implemented more effectively than ever. Companies across the world can arrange a meeting to negotiate the detail drawing of a part in the contract via the Internet using rich-media collaboration software. Contracts can be signed electronically and payments can be made instantaneously and securely.

E-Marketplaces, both of the vertical and horizontal types, can gather and match demands and supplies from all over the world. Auction processes such as Request for Quotation (RFQ) preparation, bids comparison, and order handling can be automated. Better informed and faster business decisions can be made through widely-available Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), supply chain optimization software, etc.

By operating the outsourcing processes electronically, companies can realize a number of benefits that were not feasible before. Electronic outsourcing shortens the sourcing cycle

time, reduces the time-to-market, and supports fast and informed supply-chain decision making. In terms of monetary benefit, Aberdeen Group, Inc. [1] reports that electronic outsourcing can save businesses more than \$1.7 trillion on a global basis.

In this research, we focus on auctions as trade mechanisms for outsourcing.

1.4 Multi-Attribute Outsourcing

As mentioned in Section 1.2, price is often the only decision factor considered in outsourcing decisions. If the objective of an outsourcing is to maximize the *value* delivered, proper evaluation of non-price factors are needed.

Even though the mainstream of the business world is not quite ready for the multi-attribute trade, there are important developments in some big companies. At General Electric Company (GE), a manufacturer of jet engines, price-only auctions are traditionally used for the sourcing of non-critical materials (See the MIT Center for Transportation and Logistics Newsletter and Calendar [72]). With multi-attribute auctions, GE can outsource complex parts by allowing suppliers to offer price-quality bids from which GE can choose the bid that yields the lowest cost while making no sacrifices in quality.

1.5 Price-Delivery Date Outsourcing

For manufacturing companies, in addition to price, delivery lead time is an important consideration in supplier selection. Manufacturing activities have to be coordinated with respect to the production plan. Further, the company has to decide how much to produce, and *when* to produce it. For instance, the decisions as to when the raw material should be delivered to the manufacturing facility and when the production of a particular product should start are crucial in keeping production proceeding smoothly and efficiently. In this research, delivery date, in addition to price, is considered as the second attribute in decision making associated with selecting a supplier.

1.6 Organization of Document

This document is divided into four parts. In this chapter, we have introduced the basics of outsourcing and point out the possibility and benefits of considering multi-attribute outsourcing. In Chapter 2, we present the overall research objectives, then describe the research problems. A summary of research contributions and accomplishments is also provided.

In Part II of the document, we present the auction design for multi-attribute outsourcing in manufacturing supply networks. In particular, we consider price and delivery date as the two decision factors in supplier selection. An outcome of the auction determines which item(s) are given to each supplier, the items' delivery dates, and the prices the OEM pays the supplier for supplying the items. Chapter 3 briefly discusses the background of auctions and some of the important literature in the field. We propose an auction mechanism and prove its properties in Chapter 4.

The auction mechanism presented in Part II requires the suppliers' knowledge of their costs. Part III of the document presents the supplier's cost models and algorithms for determining the cost quickly and accurately. The cost determination models that we consider are variants of lot-sizing models. In Chapter 5, therefore, we provide literature survey on the lot-sizing problems. Chapter 6 proposes heuristic algorithms and presents computational results for the capacitated lot-sizing problems with backorders (CMB). Chapter 7 proposes an MIP-based heuristic that helps to solve the capacitated lot-sizing problems with backorders and integral unit production (CMBI). Computational results are also provided. Lastly, in Part IV, we suggest possible future research directions.

Chapter 2

Research Summary

2.1 Chapter Overview

This chapter provides a summary of this research. In Section 2.2, we briefly define the model that we consider. Section 2.3 identifies the overall research objectives. Section 2.4 presents the research contributions and major accomplishments.

2.2 A Brief Model Description

We consider a multi-attribute sourcing economy consisting of an Original Equipment Manufacturer (OEM) and a set of suppliers. Price and delivery date are the two decision factors that influence the OEM's decision on supplier selection. The OEM demands a set of indivisible items, each of which has to be delivered on an allowable delivery date. We define a *tuple* as a bundle of items and the associated delivery dates of each item in the bundle. The OEM has determined private valuations on all possible tuples. Similarly, in order to participate in the auction, each supplier has to know his costs for all possible tuples.

This research addresses two problems. First, it addresses the problem of how to assign tuples to suppliers and how to determine the prices of the tuples' such that all agents, the OEM and the suppliers, are satisfied.

Second, since the suppliers are required to know their costs on tuples, this research also addresses the problem of how a supplier can quickly and accurately determine the optimal production plan that minimizes his cost on a tuple. The fundamental problem for a supplier is how to balance the effects of setup costs, inventory costs, and possibly tardiness penalties.

The supplier can model this as an optimization problem. However, these problems are combinatorial in nature and computationally intensive. Therefore, a supplier needs solution methods that are fast in finding a good solution.

2.3 Research Objectives

Our research is divided into two main parts. Their corresponding objectives are:

Objective 1 To design a mechanism that facilitates a multi-item multi-attribute outsourcing economy and achieves (approximate) *efficient assignments at competitive equilibrium* outcomes.

Objective 2 To develop solution methods that enable a supplier to determine quickly and accurately the production plan that minimizes the total cost for a tuple.

2.4 Summary of Research Contributions

In this section, we summarize the contributions of the research that is described in detail in Part II and Part III of this dissertation.

2.4.1 Auction Design

We design an auction mechanism for a multi-item multi-attribute outsourcing economies. Even though there are mechanisms for multi-item economy, to the best of our knowledge, the mechanism that we propose is the first that allows the multi-attribute feature.

In this research, we first show that there always exists a competitive equilibrium in the multi-item multi-attribute outsourcing economy, when prices are non-linear and non-anonymous. Then an auction algorithm is designed. After that, we show that the auction terminates with an approximate efficient assignment at competitive equilibrium. We also prove an inefficiency bound.

2.4.2 Cost Determination Models and Algorithms

We develop two MIP models for the capacitated lot-sizing problems with backorders. The first model, called CMB, allows production levels to vary in each period. The second model, called CMBI, has the additional restriction that the production levels have to be integral multiples of a given batch size.

We develop efficient solution methods for the two models. For CMB, we propose three valid inequality selection algorithms. These three procedures are shown to be efficient. Based on our experiments, they outperform a commercial solver in terms of computing time.

For CMBI, we propose an MIP-based heuristic. Our experiments show that the heuristic usually finds good solutions faster than a commercial solver. Moreover, this heuristic sometimes achieves optimal solutions.

Part II

Auction Design

Chapter 3

Background on Auctions

3.1 Chapter Overview

This chapter presents the motivation for the use of auction as a trade mechanism and summarizes the literature in this field that relates to our research. Section 3.2 briefly describes the concepts of *mechanism design* and *auctions*. Section 3.3 introduces the four common forms of *single-item auctions*. Section 3.4 investigates the *multi-item auctions*. In Section 3.5, a different setting of auctions, called *reverse auctions*, is described. Our research deals with the outsourcing (procurement) environments, to which the reverse auctions is relevant.

3.2 Mechanism Design and Auctions

Mechanism design, also known as *implementation problem*, involves the design of mechanisms that implement desirable social outcomes for the society's members, called agents, who have private information about their preferences over the set of the social outcomes. Examples of applications of mechanism design include *monopolistic price discrimination*, *optimal taxation*, and the design of auctions. Elaborated examples of applications of mechanism design can be found in Fudenberg and Tirole [38] and Rosenschein and Zlotkin [83]. Varian [93] is a good introductory paper to the field of mechanism design, while Mas-Colell et al. [69] provides more in-depth technical contents, suitable for those who are seriously interested in the field.

Auction is a resource allocation and price discovery mechanism. As a mechanism, auction elicits information, in the form of bids, from participants (bidders), and based on the bids received, the allocation and the associated price are determined.

Even though auctions have been used for thousands of years, they are still a primary trading institution nowadays. There are many reasons for this. With simple rules, auctions can resolve complex multi-lateral bargaining problems. Auctions, by nature, promote competition. Therefore, to be more competitive, bidders put more effort in making themselves more efficient. Competitive environment also makes auctions robust. Though rules of auctions are simple, but they are designed for the system to work well even when bidders are not completely rational.

Compared to other trade mechanisms, such as bargaining or fixed-price policy, auction is preferred in many situations. Bulow and Klemperer [21] show that auction is better than bargaining, from seller's perspective. The logic is as follows. Since seller wants to sell item at high price, and in bargaining, it is tempted to threaten not to sell the item if the price is not high enough. The act by the seller is equivalent to turning to another bidder in the auction. The only difference is that the threat creates fake competition, whereas in auction, competition is real. Needless to say, real competition is better than the fake one. Auctions usually outperform fixed-price policy as well. It is easy for the seller to misjudge the bidders' values. While overestimating bidders' values means the item might not be sold, underestimating bidders' values means lower profit. McAfee and McMillan [71] discuss in greater detail why auctions are usually preferred over other market institutions.

Varian [94] gives a simple but yet intuitive introduction to the field of auction. To name a few, good auction survey papers include Wilson [99] and McAfee and McMillan [70]. More advanced material can be found in Mas-Colell et al. [69], Krishna [54], and Fudenberg and Tirole [38].

3.3 Single-Item Auctions

The simplest auction setting is the single-item auction. Four common forms of single-item auction are the *English* auction, the *Dutch* auction, the *first-price sealed-bid* auction, and the *second-price sealed-bid* auction.

English Auctions

The English auction is the most used form of auctions. In English auction, price starts from the reserve price set by the auctioneer and increases as bidders outbid each other. Each time a bidder outbids others the price increases by a minimum price increment which is set beforehand. The item is awarded to the bidder who offers the highest price.

Dutch Auctions

The Dutch auction is the opposite of the English auction. In Dutch auction, price starts from a high price and decrease in steps (or continuously) until there is a bidder who accepts the price and the accepted bidder is awarded the item and he pays the price at which he accepted the offer. This form of auction receives its name from the flower markets of the Netherlands where is still used.

The first two forms of auctions can be viewed as iterative mechanisms. In English auction, assuming that the bidders are profit-maximizing agents, high-value bidders always have a chance to outbid lower-value bidders, therefore the allocation tends to be *efficient*, in a sense that one of the highest-value bidders eventually wins. This is not the case for the Dutch auction, where high-value bidders can be too greedy and wait too long until some lower-value bidder takes the price and wins the auction. The English auction is also susceptible for collusion. But measures such as making bidders bid in pre-specified increments and making bids anonymous can partly alleviate the problem.

First-Price Sealed-Bid Auctions

The first-price sealed bid auction is usually used for construction projects. Each of the construction companies (bidders) submits a private bid (not seen by other bidders) to the project owner. The lowest-bid company wins the construction contract and pays his bid price.

Second-Price Sealed-Bid Auctions

The fourth single-item auction form is the second-price sealed-bid auction, also known as the *Vickrey* auction. As a sealed-bid auction, each of the bidders in the Vickrey auction submits a private bid to the seller. The highest bidder gets the item, but pays the second-highest bid price.

Sealed-bid auctions, in some respect, are preferable to iterative auctions, especially the English auction. In the sealed-bid auction, bidders submit bids once and simultaneously, therefore it is harder than the English auction for bidders to collude. Furthermore, in sealed-bid auctions, unlike in the English auction, lower-value bidders are not discouraged to participate, because they have a chance to win no matter what their values are. But the side-effect of this is that the allocation can be *inefficient*, in a sense that, a lower-value bidder gets the item instead of the highest-value bidder.

Klemperer [53] thoroughly discusses in a non-technical manner the strengths and weaknesses of different auction forms. The discussion is important for the design of auctions.

3.4 Multi-Item Auctions

The analysis of the multi-item auctions is much more complicated than that of the single-item auctions. In this section, we discuss two types multi-item auctions: *direct mechanisms*, presented in Section 3.4.1, where buyers submit their *types* directly to the mechanism, and *indirect mechanisms*, presented in Section 3.4.2, where buyers indirectly reveal their *types* through iterative process.

3.4.1 Vickrey-Clarke-Groves Mechanism

The Vickrey auction is a special case of a class of mechanisms called *Vickrey-Clarke-Grove* mechanisms (VCG), proposed by Vickrey [95], Clarke [26], and Grove [40]. In VCG mechanism, each buyer reports its type (valuations) which may or may not be truthful, and based on the reported types, the mechanism computes an efficient allocation (valuation-maximizing allocation) and the Vickrey payments (buyers' payments to the mechanism). Let $V(\mathcal{N})$ and $V(\mathcal{N}\setminus j)$ denote the values of the efficient allocations when *all* the buyers in \mathcal{N} participate in the auction and when all but buyer j participate, respectively. Buyer j 's *marginal product*, MP_j , is defined by:

$$MP_j = V(\mathcal{N}) - V(\mathcal{N}\setminus j)$$

Suppose in an efficient allocation buyer j receives bundle \mathcal{S} , then buyer j 's Vickrey payment, VP_j , is the sum of $v_j(\mathcal{S})$ and MP_j , where $v_j(\mathcal{S})$ is buyer j 's valuation on bundle \mathcal{S} . The VCG mechanism for buyers with quasi-linear utility function is *incentive compatible* – truthful bidding, i.e., reporting the value truthfully, is a weakly-dominant strategy for all the bidders. The benefit of this property is that buyers do not have to speculate on other bidders' strategies, but yet the efficient outcome can be achieved. Following is an example for a special case where each buyer demands at most one item.

Example 1 Buyers 1 and 2 are interested in buying either item A or B. The following table shows the buyers' valuations on the items.

	Item A	Item B
Buyer 1	10	6✓
Buyer 2	9✓	3

As indicated by the sign “✓,” the efficient allocation is to assign item A to buyer 2 and assign item B to buyer 1, with the total value of 15. For marginal products, MP_1 is 6 ($= 15 - 9$) and MP_2 is 5 ($= 15 - 10$). The Vickrey payments VP_1 and VP_2 therefore are 0 and 4, respectively.

The VCG mechanism can be implemented for a variety of problems. Leonard [59] considers the problem of matching individuals with positions. An individual has preferences (values) over the positions and will be charged if assigned to a position. An efficient assignment is an assignment that maximizes the total values. Let \mathbf{p} denotes a price vector that supports efficient assignment. We say that \mathbf{p} is the *minimum price equilibrium* if there is no other price vector \mathbf{p}' at an equilibrium such that a component in \mathbf{p}' is less than the corresponding component in \mathbf{p} . Using the primal-dual analysis of the LP for assignment problem¹, a direct mechanism which computes the minimum price equilibrium is developed. Each individual, under such mechanisms, receives her marginal product at equilibrium and therefore has no incentive to be dishonest in reporting their values. The individual-position matching mechanism is incentive compatible and can be considered a generalization of the Vickrey auction.

For multi-item trade involving multiple buyers and multiple sellers, each with multiple-unit demand, Bikhchandani and Ostroy [16] study the interconnections among the linear programming (LP) formulation of the model, Walrasian (pricing) equilibrium, and the core. Four orders of pricing equilibria (PE) are derived through four primal-dual LP formulations, primal defines efficient allocation and dual defines equilibrium prices.

First-order PE: At equilibrium, prices are *anonymous*, i.e., the price of an item is the same for all buyers, and also *linear*, i.e., the price of a bundle equals the sum of the prices of the items in it.

Second-order PE: At equilibrium, prices are *anonymous*, and also *non-linear*, i.e., the price of a bundle is not necessarily equal to the sum of the prices of the items in it.

Third-order PE: At equilibrium, prices are *non-anonymous* with respect to buyers, i.e., the price of a bundle depends on identity of the buyer, and also *non-linear*.

¹Shapley and Shubik[87] are the first who analyze the assignment game using primal-dual formulations analysis. They find that the outcomes in the core of the game are the solutions of the dual to the efficient assignment problem. Moreover, these dual solutions correspond to the equilibrium prices.

Fourth-order PE: At equilibrium, prices are *non-anonymous* with respect to both sellers and buyers, i.e., the price of a bundle depends on identity of the buyer and the seller, and also *non-linear*.

For a single seller case, Bikhchandani and Ostroy show that equilibria with non-linear, non-anonymous prices exist and coincide with the core. They also show how buyers' Vickrey payments can be determined, given that the following necessary and sufficient condition, is satisfied:

$$V(\mathcal{N}) - V(\mathcal{K}) \geq \sum_{j \in \mathcal{N} \setminus \mathcal{K}} [V(\mathcal{N}) - V(\mathcal{N} \setminus j)], \quad \forall \mathcal{K} \subseteq \mathcal{N}$$

which is interpreted to mean *buyers are substitutes*. In words, this condition states that the marginal product of a group of buyers is more than the some of the marginal products of the individual buyers of the group. In Bikhchandani and Ostroy [16], to determine the Vickrey payments, two optimization problems have to be solved, primal and dual. Bikhchandani et al. [15] proposes a new formulation such that Vickrey payments can be determine by solving only one optimization problem.

Gul and Stacchetti [41] study the problem similar to Bikhchandani and Ostroy [16]'s, but their focus of investigation is the existence of the first-order PE. In this regard, two new conditions necessary for the existence of the first-order PE are proposed. The authors also prove that the Vickrey prices are less than or equal to the smallest Walrasian equilibrium prices.

There are two obvious concerns for the VCG implementations proposed by Bikhchandani and Ostroy [16] and Bikhchandani et al. [15]. First, in order to determine Vickrey allocation and prices, we have to solve optimization problem(s), which are *NP-hard*. Second, even though the VCG mechanisms are incentive compatible, by nature, buyers are not comfortable revealing their true valuations to the mechanism directly. In Sections 3.4.2, we introduce an alternative approach in which buyers indirectly reveal their types through iterative price-increasing processes. These indirect mechanisms converge to approximated equilibria where

the convergence speed depends on the size of the bid increment. Furthermore, in these indirect mechanisms, buyers do not need to reveal their values.

3.4.2 Iterative Mechanisms

Bertsekas [11, 12, 13] introduce the distributed relaxation algorithm for the assignment problem, called the AUCTION algorithm. Certainly, the algorithm operates like an auction which matches N individuals to N items. Each individual has private information about his valuations over the items. The algorithm assigns individuals to items such that the efficient assignment, i.e., a value-maximizing assignment, is obtained. There are two phases in each iteration of the algorithm, the *bidding phase* and the *assignment phase*. In the bidding phase, each individual j that is not assigned to any item under current assignment finds the items i^* and i' that give highest profit $\tilde{\pi}_{i^*j}$ and second-highest profit $\tilde{\pi}_{i'j}$. Individual j 's bid for item i^* , denoted by b_{i^*j} , is:

$$b_{i^*j} = p_{i^*} + \tilde{\pi}_{i^*j} - \tilde{\pi}_{i'j} + \epsilon$$

where p_{i^*} and ϵ are the current price of item i^* and the bid increment, respectively. In the assignment phase, based on the bids received, the price of an item i is set to the maximum bid price.

$$p_i = \max_{j \in \mathcal{J}_i} b_{ij}$$

where \mathcal{J}_i is the set of individuals who bid on item i . The assignment can be found by assigning item i to individual j that attains p_i above. The notion of ϵ -complementary slackness is introduced, and based on it, the algorithm is proved to terminate with the value within $N\epsilon$ from optimum.

Other iterative mechanisms for the assignment problem include those of Demange et al. [28]. Recall that the Leonard [59]'s direct mechanism for the assignment problem computes the minimum price equilibrium and therefore no individual has incentive to misrepresent his valuations. Demange et al. [28] study the same problem as Leonard [59]'s. They develop two iterative mechanisms, one achieves the exact minimum price equilibrium another achieves the approximated one.

Kelso and Crawford [50] investigate a situation in labor markets where firms consider hiring workers. Each firm can hire more than one worker, but a worker can work at most with one firm. Each worker has private information about his minimum salary required from each firm. The worker's utility is defined by the difference between the actual salary received (to be determined by the mechanism) and the minimum salary required. Each firm has private information about their gross product, given a group of workers are hired. The firm's utility is defined by the gross product of the group minus the sum of the salary paid to each worker in the group. Obviously, the assignment problem is a special case of this problem. Kelso and Crawford [50] generalize the result of Shapley and Shubik [87] by showing that Walrasian equilibrium in such model exists, provided that all workers are *gross substitutes* from each firm's standpoint. They also develop the salary-adjustment process that converges to the equilibrium biased in favor of agents on the side of the market that make offers.

For exchange economy in which each buyer is allowed to buy a bundle of items, Parkes [79] develop a primal-dual-based iterative algorithm, called COMBAUCTION. The algorithm relies on the primal-dual formulations developed by Bikhchandani and Ostroy [16]. Three variants of COMBAUCTION are proposed. The variant called COMBAUCTION(2) keeps prices anonymous throughout the process, while the COMBAUCTION(3) variant allows non-anonymous prices. In COMBAUCTION(D), the last variant, prices are anonymous at first (through COMBAUCTION(2)) until a certain condition fails from which point prices are made non-anonymous (through COMBAUCTION(3)). There is an observation that should be mentioned. The variants COMBAUCTION(2) and COMBAUCTION(D) do not always achieve the second-order PE, even in the case where the second-order PE actually exists. But they always terminate at least at a third-order PE. Among other results, Parkes [79] show that inefficiency can occur because of the discreteness of the price increment. He also proves the bound for the inefficiency.

3.5 Reverse Auctions

Most of the auctions that we discussed so far are the auctions with a single seller who offers to sell items to multiple buyers. In the outsourcing scenario that we consider, however, there is a single buyer (OEM) who wants to buy items² from multiple sellers (suppliers). The auctions for such outsourcing scenario are called the *reverse auction*. The allocation and payment rules as well as properties of the auctions discussed so far can be easily adapted for reverse auctions. Consider the following example of the adaptation of the VCG mechanism in the form of reverse auction for an outsourcing situation.

Example 2 An OEM wants to procure item A and B. There are two suppliers having the capability to produce them, suppliers 1 and 2. However, due to capacity constraint, at the moment suppliers 1 and 2 can produce at most one item. The following table shows the OEM's valuations and suppliers' costs on the items.

	Item A	Item B
OEM's valuations	15	15
Supplier 1's costs	10✓	6
Supplier 2's costs	9	3✓

As indicated by the sign “✓,” the efficient allocation is to let supplier 1 supply item A and supplier 2 supply item B. This allocation gives the value of 17 ($= 5 + 12$). For marginal products, MP_1 is 5 ($= 17 - 12$) and MP_2 is 8 ($= 17 - 9$). The Vickrey payments VP_1 and VP_2 therefore are 15 ($= 10 + 5$) and 11 ($= 3 + 8$), respectively. Due to the allocation and payment rules shown in this example, the suppliers have no incentive to misrepresent their costs.

²We can think of the items as products or services.

Chapter 4

Proposed Auction Algorithm

4.1 Chapter Overview

This chapter proposes an auction algorithm that achieves approximate efficient assignment in competitive equilibrium. Section 4.2 describes the outsourcing economy. Section 4.3 formulates the outsourcing model as linear programs. Section 4.4 provides the price interpretation from the linear programs. In Section 4.5, we show that competitive equilibrium always exists in the outsourcing economy. Section 4.6 proposes the auction algorithm and proves its the optimality properties.

4.2 The Model

We consider an outsourcing economy consisting of an OEM and a set of suppliers, $\mathcal{N} = \{1, \dots, N\}$. The OEM has demand for a set of indivisible items $\mathcal{I} = \{1, \dots, I\}$. Every item i in \mathcal{I} has to be delivered on a date k in set $\mathcal{D} = \{1, \dots, D\}$. A *bundle* \mathcal{S} is a set of item(s) where $\mathcal{S} \subseteq \mathcal{I}$.

Definition 4.1 (Tuple) A “tuple”, τ , is defined by:

$$\tau = (\mathcal{S}, (i, k_i) : \forall i \in \mathcal{S})$$

where $\mathcal{S} \subseteq \mathcal{I}$ and k_i is the delivery date of item i .

For example, let $\mathcal{N} = \{1, 2\}$, $\mathcal{I} = \{A, B\}$, $\mathcal{D} = \{Mon, Tue\}$. The possible tuples include $\tau_1 = (\{B\}, (B, Wed))$, $\tau_2 = (\{A, B\}, (A, Mon), (B, Tue))$, $\tau_3 = (\{A, B\}, (A, Tue), (B, Tue))$, and so on.

In cases where more attributes are required for decision-making, we can define tuple τ as:

$$\tau = (\mathcal{S}, (i, k_i, \dots, l_i) : \forall i \in \mathcal{S})$$

where k_i, \dots, l_i represent attributes' values of item i in \mathcal{S} . The results that we present can be easily applied to such cases as well.

The OEM has a private valuation $v(\tau)$ on every tuple (τ). The valuations are assumed non-negative. Similarly, for every tuple (τ), supplier j privately knows his production cost, which is denoted by $c_j(\tau)$. Chapters 6 and 7 address the supplier's problem of determining the minimum production cost $c_j(\tau)$. In those chapters, we develop valid inequalities, the valid inequality selection procedures, as well as an MIP-based heuristic in order to determine $c_j(\tau)$ quickly and accurately. Again, the costs are assumed non-negative. In the case where a tuple is not desirable to the OEM (e.g., a bundle is definitely not desired on a particular delivery date for some reason), the OEM can set the tuple's valuation to zero. Similarly, a supplier may not be able to produce and deliver some tuple, in which case, the supplier can set the production cost of the tuple to infinity.

Definition 4.2 (Assignment) An “assignment”, represented by a vector

$$\alpha = \{(1, \tau_1), \dots, (N, \tau_N)\}$$

is an indication as to which supplier is assigned to which tuple.

Definition 4.3 (Feasible Assignment) Let \mathcal{S}_j be the bundle in tuple τ_j . An assignment $\alpha = \{(1, \tau_1), \dots, (N, \tau_N)\}$ is a “feasible assignment” if:

- a) All the bundles \mathcal{S}_j for all $j \in \mathcal{N}$ are subsets of \mathcal{I}
- b) All the delivery dates of the items in \mathcal{S}_j for all $j \in \mathcal{N}$ are in \mathcal{D}
- c) None of the items is assigned to multiple suppliers: $\mathcal{S}_j \cap \mathcal{S}_l = \emptyset$, for all $j \neq l$

We denote a set of all possible feasible assignments by \mathcal{A} . Note that an *empty bundle*, containing none of the items in I , is also a subset of \mathcal{I} . Therefore, an assignment in which

some of the suppliers are assigned to empty bundles can still be a feasible assignment if conditions b) and c) are satisfied. In this context, an assignment in which all of the suppliers are assigned to empty bundles is also considered a feasible assignment.

A tuple with empty bundle is called an *empty tuple*, and it is denoted by $\tau(\emptyset)$. An assignment in which all of the suppliers are assigned to empty tuples, is called an *empty assignment*, and it is denoted by $\alpha(\emptyset)$. Let $p(\tau)$ be the price of a tuple τ .

The OEM wants to maximize his utility, or surplus, which is defined by:

$$U_c(\alpha) = \sum_{\tau \in \alpha} v(\tau) - p(\tau) \quad (4.1)$$

Each supplier j in \mathcal{N} wants to maximize utility (profit) which is defined by:

$$U_j(\tau) = p(\tau) - c_j(\tau) \quad (4.2)$$

We assume that:

- Consumer's valuation for the empty tuple is zero: $v(\tau(\emptyset)) = 0$
- Producer's cost of producing an empty tuple is zero: $c_j(\tau(\emptyset)) = 0$, for all $j \in \mathcal{N}$

The two assumptions are realistic. The second assumption complies with most of the generic production models, including our lot-sizing models, namely CMB and CMBI, which are presented in chapters 6 and 7, respectively.

It is easy to show that the price of an empty tuple is also zero, i.e., $p(\tau(\emptyset)) = 0$. Therefore, from equation (4.1), we can see that the OEM's surplus for an empty tuple assignment is zero, i.e., $U_c(\alpha(\emptyset)) = 0$. Since both the OEM and the suppliers are utility-maximizing agents, they would prefer zero utility to negative utility. For this reason, the OEM prefers an empty tuple assignment, which gives zero utility, to non-empty tuple assignments that give negative utility. Similarly, the suppliers prefer empty tuples, which give zero utility, to non-empty tuples that give negative utility.

Definition 4.4 (Economy Surplus) An “economy surplus” is the sum of the utility of all the agents in the economy.

We denote the economy described above by $\mathcal{E}(\mathcal{N}, \mathcal{I}, \mathcal{D}, U_c, U_j)$, or simply \mathcal{E} . Our objective is to find a stable outcome of \mathcal{E} such that the economy surplus is maximized.

4.3 Linear Programming Formulations

Given the utility functions of the OEM and the suppliers, shown in (4.1) and (4.2), and a feasible assignment $\alpha = \{(1, \tau_1), \dots, (N, \tau_N)\}$, the economy surplus can be written as:

$$U_{\mathcal{E}}(\alpha) = \sum_{(j, \tau_j) \in \alpha} [v(\tau_j) - p(\tau_j)] + \sum_{j \in \mathcal{N}} [p(\tau_j) - c_j(\tau_j)] \quad (4.3)$$

$$= \sum_{(j, \tau_j) \in \alpha} [v(\tau_j) - c_j(\tau_j)] \quad (4.4)$$

Note that the economy surplus is not determined by prices. Instead, it is determined by the valuations and the costs of the assigned tuples. Though prices play no role in determining the total utility of the economy, they play a key role in allocating the surplus between the OEM and the suppliers.

Definition 4.5 (Efficient Assignment) A feasible assignment is said to be “efficient” if it maximizes the economy surplus, $U_{\mathcal{E}}$, over all possible assignments.

An outcome of the economy describes who gets which item(s), if at all, and at what price. In this section we will develop LP formulations that determine an efficient assignment and find the prices that support the efficient assignment.

Since the decision that we have to make is which tuples should be assigned to which suppliers, it is natural to formulate the economy in Integer Program (IP), using binary variables representing the decisions. We now define the decision variable:

$$x_j(\tau) = \begin{cases} 1, & \text{if a non-empty tuple } \tau \text{ is assigned to supplier } j \\ 0, & \text{otherwise} \end{cases} \quad (4.5)$$

Following is a straightforward IP interpretation of \mathcal{E} .

$$(\mathbf{IP1}) \quad V_{IP1} = \max \sum_{j \in \mathcal{N}} \sum_{\tau \in \mathcal{T}} [v(\tau) - c_j(\tau)] \cdot x_j(\tau) \quad (4.6)$$

subject to

$$\sum_{\tau \in \mathcal{T}} x_j(\tau) \leq 1, \quad \forall j \in \mathcal{N} \quad (4.7)$$

$$\sum_{j \in \mathcal{N}} \sum_{\tau \ni i} x_j(\tau) \leq 1, \quad \forall i \in \mathcal{I} \quad (4.8)$$

$$x_j(\tau) \in \{0, 1\}, \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.9)$$

The objective (4.6) is to maximize the economy surplus. It is easy to see that this objective function is equivalent to the total utility function, $U_{\mathcal{E}}$, presented in (4.4). Constraints (4.7) ensure that each supplier is assigned to at most one tuple. Constraints (4.8) restrict that an item can be assigned to at most one supplier. This means that, in an efficient assignment, some item(s) are possibly not assigned to any suppliers at all. We use the notation $\tau \ni i$ in (4.8) to represent “for all tuples that contain item i ” condition. Constraints (4.9) specify that assignment variables are binary. Constraints (4.7), (4.8), and (4.9) define the feasible assignment described in Definition 4.3.

The LP relaxation of IP1 does not always yield integral optimal solution. To see this, consider the following example.

Example 3 The OEM demands two items, A and B, from two suppliers, S1 and S2. Both of the items must be delivered on one delivery date. The following table shows OEM’s valuations and suppliers’ costs on the bundles.

	{A}	{B}	{A, B}
Valuation	4	4	10
S1’s cost	4	4	7*
S2’s cost	2	2	9

Obviously, the efficient assignment is to assign bundle $\{A, B\}$ to S1, by which the objective value, V_{IP1} , is 3. Solving this problem through the LP relaxation of IP1 gives the

objective value, V_{LP1} , of 3.5. Since V_{IP1} is not equal to V_{LP1} , we know that LP1's optimal solution is not always integral.

The rest of this section is devoted to the construction of an LP that always gives the integral optimal solution.

The fact that the economy can be formulated as an LP, and that the LP always gives an integral optimal solution is crucial. In section 4.5, by employing the integral property of the LP solution, we prove the existence of the competitive equilibrium in the economy.

We now introduce a new variable, $y(\alpha)$, which indicates whether an assignment α is selected.

$$y(\alpha) = \begin{cases} 1, & \text{if an assignment } \alpha \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$$

A new integer program, called IP2, is developed. The IP2 formulation is equivalent to the IP1 formulation, in a sense that it makes the same decisions and gives the same objective values as what IP1 does.

$$\text{(IP2)} \quad V_{IP2} = \max \sum_{j \in \mathcal{N}} \sum_{\tau \in \mathcal{T}} [v(\tau) - c_j(\tau)] \cdot x_j(\tau) \quad (4.10)$$

subject to

$$\sum_{\alpha \in \mathcal{A}} y(\alpha) = 1 \quad (4.11)$$

$$x_j(\tau) \leq \sum_{\alpha \ni (j, \tau)} y(\alpha), \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.12)$$

$$\sum_{\tau \in \mathcal{T}} x_j(\tau) \leq 1, \quad \forall j \in \mathcal{N} \quad (4.13)$$

$$x_j(\tau) \in \{0, 1\}, \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.14)$$

$$y(\alpha) \in \{0, 1\}, \quad \forall \alpha \in \mathcal{A} \quad (4.15)$$

The objective (4.10), again, is to maximize the economy surplus. Constraints (4.11) states that one assignment (possibly empty) has to be selected. In constraints (4.12), we write

$\alpha \ni (j, \tau)$ to mean “for all assignments that assign tuple τ to j .” Through this set of constraints, we make sure that a tuple can be assigned to a particular supplier only if the selected assignment allows. We can set these constraints to equalities without changing the optimal solutions. However, setting the constraints to inequalities is useful in economic interpretation of a dual variable. Constraints (4.13) is in fact redundant. But, as will be clear in Section 4.5, they are necessary in developing the dual that gives useful information. Constraints (4.14) and (4.15) restrict the variables to be binary.

Obviously, IP2 is larger than IP1 in terms of number of rows and columns. This is because in IP2, we introduce a new variable, $y(\alpha)$. Moreover, the number of constraints in IP2, specifically constraints (4.12), grows exponentially with the size of \mathcal{I} .

Let LP2 be the LP relaxation of IP2. Then, we have:

$$\text{(LP2)} \quad V_{LP2} = \max \sum_{j \in \mathcal{N}} \sum_{\tau \in \mathcal{T}} [v(\tau) - c_j(\tau)] \cdot x_j(\tau) \quad (4.16)$$

subject to

$$\sum_{\alpha \in \mathcal{A}} y(\alpha) = 1 \quad (4.17)$$

$$x_j(\tau) \leq \sum_{\alpha \ni (j, \tau)} y(\alpha), \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.18)$$

$$\sum_{\tau \in \mathcal{T}} x_j(\tau) \leq 1, \quad \forall j \in \mathcal{N} \quad (4.19)$$

$$x_j(\tau) \geq 0, \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.20)$$

$$y(\alpha) \geq 0, \quad \forall \alpha \in \mathcal{A} \quad (4.21)$$

Let $(\mathbf{x}, \mathbf{y}) = ((x_j(\tau)), (y(\alpha)))$ and $(\mathbf{x}^*, \mathbf{y}^*) = ((x_j^*(\tau)), (y^*(\alpha)))$ denote a feasible and an optimal solutions, respectively.

Theorem 4.1 LP2 gives integral $(\mathbf{x}^*, \mathbf{y}^*)$.

Proof: Let H be the polyhedron defined by constraints (4.17)–(4.21) and let $(\mathbf{x}', \mathbf{y}')$ be an extreme point of H . To prove the theorem, we first show that $(\mathbf{x}', \mathbf{y}')$ is integer in \mathbf{y}' . Then, we show that if $(\mathbf{x}', \mathbf{y}')$ is integer in \mathbf{y}' , it is also integer in \mathbf{x}' .

Let $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ be a point in H that is not integer in $\bar{\mathbf{y}}$, with $\bar{y}(\alpha_1), \dots, \bar{y}(\alpha_r) > 0$. We define additional r feasible integer points $(\underline{\mathbf{x}}(\alpha_s), \underline{\mathbf{y}}(\alpha_s))$, $s = 1, \dots, r$, where,

$$\underline{\mathbf{x}}(\alpha_s) = (x_j(\tau) \mid x_j(\tau) = 1 \leftrightarrow (j, \tau) \in \alpha_s, \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T})$$

and,

$$\underline{\mathbf{y}}(\alpha_s) = (\underline{y}(\alpha_{s'}) \mid \underline{y}(\alpha_{s'}) = 1 \leftrightarrow s' = s)$$

We can show that $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is a convex combination of the r integer points $(\underline{\mathbf{x}}(\alpha_s), \underline{\mathbf{y}}(\alpha_s))$, $s = 1, \dots, r$:

$$(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = \sum_{s=1}^r \bar{y}(\alpha_s) (\underline{\mathbf{x}}(\alpha_s), \underline{\mathbf{y}}(\alpha_s))$$

Therefore, $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is not an extreme point. In other words, every extreme point $(\mathbf{x}', \mathbf{y}')$ in H is integer in \mathbf{y}' .

Because of constraints (4.18) and since \mathbf{y}' in $(\mathbf{x}', \mathbf{y}')$ is integer, \mathbf{x}' must also be integer. In other words, every extreme point $(\mathbf{x}', \mathbf{y}')$ in H is integer. \square

To derive the dual form of LP2, we need to identify the dual variables. Let π_c , $w_j(\tau)$, and π_j denote the dual variables associated with constraints (4.17), (4.18), and (4.19), respectively. Following is the dual of LP2, denoted by DLP2.

$$\text{(DLP2)} \quad V_{DLP2} = \min \sum_{j \in \mathcal{N}} \pi_j + \pi_c \quad (4.22)$$

subject to

$$x_j(\tau) : \quad w_j(\tau) + \pi_j \geq v(\tau) - c_j(\tau), \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.23)$$

$$y(\alpha) : \quad \pi_c \geq \sum_{(j, \tau) \in \alpha} w_j(\tau), \quad \forall \alpha \in \mathcal{A} \quad (4.24)$$

$$\pi_j \geq 0, \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.25)$$

Constraints (4.23) and (4.24) are associated with LP2's variables $x_j(\tau)$ and $y(\alpha)$, respectively. DLP2 has useful economic interpretations. The variable $w_j(\tau)$ represents the OEM's surplus due to the assignment of tuple τ to supplier j . Let \mathbf{w}_j be the vector for $w_j(\tau)$ for

all τ in \mathcal{T} . The two variables π_c and π_j respectively represent OEM's maximum surplus and supplier j 's maximum profit, given \mathbf{w}_j for all j in \mathcal{N} .

Constraints (4.23) ensure that, for a tuple τ , the sum of the supplier j 's maximum profit, π_j , and the OEM's surplus, $w_j(\tau)$, is greater than or equal to the economy surplus due to the assignment of the tuple. Constraints (4.24) ensure that the OEM's maximum surplus, π_c , must be greater than or equal to the sum of each surplus $w_j(\tau)$ in any assignment. Constraints (4.25) indicate that suppliers' profits must be non-negative. A feasible solution of DLP2 is denoted by a vector $(\pi_c, (\pi_j), (\mathbf{w}_j))$.

A feasible solution to LP2 and a feasible solution to DLP2 are respectively optimal to LP2 and DLP2 if and only if the complementary slackness conditions are satisfied. For LP2-DLP2 pair, the complementary slackness conditions are:

$$[(w_j(\tau) + \pi_j) - (v(\tau) - c_j(\tau))] \cdot x_j(\tau) = 0, \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.26)$$

$$[\pi_c - \sum_{(j,\tau) \in \alpha} w_j(\tau)] \cdot y(\alpha) = 0, \quad \forall \alpha \in \mathcal{A} \quad (4.27)$$

$$[1 - \sum_{\tau \in \mathcal{T}} x_j(\tau)] \cdot \pi_j = 0, \quad \forall j \in \mathcal{N} \quad (4.28)$$

These conditions are crucial to the derivation of our results.

4.4 Prices

In section 4.3, we described LP formulations of the economy \mathcal{E} . The LP2 formulation determines efficient assignments while the DLP2 formulation determines agents' profits that support the efficient assignment. It is interesting to note that all of the variables in DLP2 can be interpreted to represent profits, either to suppliers or OEM. None of them explicitly yields the equilibrium price. However, price is typically the decision variable that gets adjusted in most trading mechanisms, including ours, in order for the mechanism to achieve the efficient assignment. In this section, we introduce price variables and show that they can be used to describe the feasible set, i.e., all of DLP2 constraints.

There are two sets of constraints in DLP2, constraints (4.23) and (4.24). First consider constraints (4.23). Rearranging them, we have:

$$c_j(\tau) + \pi_j \geq v(\tau) - w_j(\tau), \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.29)$$

The left hand side of (4.29) can be interpreted as the price of a tuple τ at which supplier j achieves his maximum profit. We call it the *bid price*, $p_j^b(\tau)$, and it is defined by:

$$p_j^b(\tau) = c_j(\tau) + \pi_j \quad (4.30)$$

We interpret the right hand side of (4.29) to represent the price for a tuple τ that the customer is willing to pay to supplier j , given $w_j(\tau)$. We call it the *offer price*, $p_j^o(\tau)$, and it is defined by:

$$p_j^o(\tau) = v(\tau) - w_j(\tau) \quad (4.31)$$

Using $p_j^b(\tau)$ and $p_j^o(\tau)$, inequalities (4.29) can be rewritten as:

$$p_j^b(\tau) \geq p_j^o(\tau), \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.32)$$

Inequalities (4.32) are intuitive because, for any tuple, while suppliers want to trade at the highest prices possible, OEM wants to trade at the lowest. Unless the price of a tuple is agreed upon by both OEM and supplier j , there will be no trade of the tuple between the two agents. We define an *agreed price*, denoted by $p_j(\tau)$, to be a price of a tuple τ at which both sides agree upon, i.e., if there exists an agreed price $p_j(\tau)$, then $p_j(\tau) = p_j^b(\tau) = p_j^o(\tau)$.

We now show how constraints (4.24) can be described using price variables. From (4.23) and (4.30), the profit w_j can be expressed as:

$$w_j(\tau) \geq v(\tau) - p_j^b(\tau), \quad \forall j \in \mathcal{N}, \forall \tau \in \mathcal{T} \quad (4.33)$$

Therefore, we can rewrite (4.24) using price variables as:

$$\pi_c \geq \sum_{(j,\tau) \in \alpha} v_j(\tau) - p_j^b(\tau), \quad \forall \alpha \in \mathcal{A} \quad (4.34)$$

Let \mathbf{p}^b and \mathbf{p}^o , respectively, be the *bid price vector* and the *offer price vector* representing prices of all possible tuples. Recall from (4.23) and (4.24) that DLP2 is defined in the $(\pi_c, (\pi_j), (\mathbf{w}_j))$ -domain. But in this section, from (4.32) and (4.34), we show that DLP2 can be recast as an equivalent problem in the $(\pi_c, \mathbf{p}^b, \mathbf{p}^o)$ -domain. Furthermore, the solutions to the two problems can be mapped back and forth very easily through (4.30) and (4.31).

The prices interpreted from DLP2 are non-linear and non-anonymous. They are non-linear because prices of items are not additive. They are non-anonymous because the price of a tuple depends on the identity of the supplier.

In section 4.3, we demonstrate that LP2 gives an optimal integral solution. Furthermore, by solving LP2 we obtain an efficient assignment, $(\mathbf{x}^*, \mathbf{y}^*)$, and by solving DLP2, we obtain an optimal profit vector, $(\pi_c^*, (\pi_j^*), (\mathbf{w}_j^*))$, that supports $(\mathbf{x}^*, \mathbf{y}^*)$. In section 4.4, we demonstrate that for a given profit vector $(\pi_c, (\pi_j), (\mathbf{w}_j))$, we can derive the corresponding price vector $(\mathbf{p}^o, \mathbf{p}^b)$, and vice versa. Therefore, an outcome of the economy, i.e., who gets which tuple and at what price, can be described by (\mathbf{x}, \mathbf{y}) and $(\pi_c, (\pi_j), (\mathbf{w}_j))$, or equivalently by (\mathbf{x}, \mathbf{y}) and $(\mathbf{p}^o, \mathbf{p}^b)$. At this point, we are ready to formally define competitive equilibrium in LP2-DLP2 context.

Definition 4.6 (Competitive Equilibrium) An outcome of an economy defined by (\mathbf{x}, \mathbf{y}) and $(\mathbf{p}^o, \mathbf{p}^b)$ is at competitive equilibrium if:

- i.) Given the price vector $(\mathbf{p}^o, \mathbf{p}^b)$, the assignment (\mathbf{x}, \mathbf{y}) maximizes the OEM's surplus.
- ii.) Given the price vector $(\mathbf{p}^o, \mathbf{p}^b)$, the assignment (\mathbf{x}, \mathbf{y}) maximizes the supplier j 's profit, for all j in \mathcal{N} .
- iii.) The market clears (demand equals supply).

4.5 Existence of Competitive Equilibrium

Competitive equilibrium is a desirable economic property for an economy. Relying on the connection between the complementary slackness conditions and the competitive equilibrium conditions, we show in this section that there exists a competitive equilibrium in \mathcal{E} .

Theorem 4.2 Let $(\mathbf{p}^{o*}, \mathbf{p}^{b*})$ be the price vector derived from $(\pi_c^*, (\pi_j^*), (\mathbf{w}_j^*))$. There exists a competitive equilibrium in \mathcal{E} defined by $(\mathbf{x}^*, \mathbf{y}^*)$ and $(\mathbf{p}^{o*}, \mathbf{p}^{b*})$.

Proof: According to the definition of the competitive equilibrium, this theorem follows from the following three lemmas. In the proof, we refer back to the complementary slackness conditions presented in (4.26)–(4.28).

Lemma 4.1 Given $(\mathbf{x}^*, \mathbf{y}^*)$ and $(\mathbf{p}^{o*}, \mathbf{p}^{b*})$, each supplier in \mathcal{E} maximizes his profit.

Proof: Through LP2, supplier j receives at most one tuple. Consider complementary slackness condition (4.26). It implies that:

$$x_j^*(\tau) = 1 \quad \rightarrow \quad w_j^*(\tau) + \pi_j^* = v_j(\tau) - c_j(\tau)$$

Therefore, if supplier j is assigned to tuple τ then:

$$\pi_j^* = [v_j(\tau) - c_j(\tau)] - w_j^*(\tau) \tag{4.35}$$

$$\geq [v(\tau') - c_j(\tau')] - w_j^*(\tau'), \quad \forall \tau' \in \mathcal{T} \tag{4.36}$$

$$= p_j^{o*}(\tau') - c_j(\tau'), \quad \forall \tau' \in \mathcal{T} \tag{4.37}$$

where inequalities (4.36) are directly implied from constraints (4.23). \square

Lemma 4.2 Given $(\mathbf{x}^*, \mathbf{y}^*)$ and $(\mathbf{p}^{o*}, \mathbf{p}^{b*})$, the OEM in \mathcal{E} maximizes his utility.

Proof: Suppose an assignment α is selected by the OEM. Complementary slackness condition (4.27) implies that:

$$y^*(\alpha) = 1 \quad \rightarrow \quad \pi_c^* = \sum_{(j,\tau) \in \alpha} w_j^*(\tau)$$

But for any other assignment α' , we know from (4.24) that:

$$\pi_c^* \geq \sum_{(j,\tau) \in \alpha'} w_j^*(\tau), \quad \forall \alpha' \in \mathcal{A}$$

$$= \sum_{(j,\tau) \in \alpha'} v_j(\tau) - p_j^{b*}(\tau), \quad \forall \alpha' \in \mathcal{A}$$

Clearly, the OEM selects assignment α which maximizes his utility. \square

Lemma 4.3 In \mathcal{E} , the market always clears.

Proof: Formulation LP2 assigns tuples to suppliers, in other words, it matches demand and supply, through the assignment variable, $x(\cdot)$. As a result, there is no excess demand or supply. Therefore, the market in \mathcal{E} always clears. \square

The theorem follows from the three Lemmas above. \square

4.6 Iterative Auction Algorithm

Given the valuation vector, \mathbf{v} , and the cost vectors, \mathbf{c}_j of all supplier j in \mathcal{N} , a competitive equilibrium in \mathcal{E} can be achieved by directly solving LP2-DLP2. However, there are two difficulties in such a direct mechanism. First, the number of variables and constraints in LP2-DLP2 grows exponentially with the size of \mathcal{I} . Second, agents in the economy are not comfortable revealing their private information such as cost to others.

In this section, we develop a price-decreasing primal-dual-based mechanism for the LP2-DLP2 pair. The connection between the LP formulations and the algorithm's outputs are presented. Then, we prove the properties of the algorithm.

4.6.1 Proposed Algorithm

The values of LP2 and DLP2 are denoted by V_{LP2} and V_{DLP2} , respectively. Similarly, the optimal values of LP2 and DLP2 are denoted by V_{LP}^* and V_{DLP2}^* , respectively. According to the duality theorem, if both LP2 and DLP2 are feasible, we obtain the following relationship:

$$V_{LP2} \leq V_{LP2}^* = V_{DLP2}^* \leq V_{DLP2}$$

Given a primal feasible solution (i.e., a feasible assignment) and a dual feasible solution (i.e., a feasible profit vector), our algorithm iteratively updates prices by which the gap between V_{LP2} and V_{DLP2} is narrowed down. We assume that the price decrement, denoted by ϵ , is positive and constant. Let V_{OUT} be the value when the algorithm terminates. In

this section, we show a bound for the discrepancy between V_{OUT} and V_{LP2}^* (or, equivalently, V_{DLP2}^*) that depends on ϵ .

In each iteration of **OUTSOURCING**, we need to know who is assigned to which tuple, and what are the offer prices and the bid prices at the moment. Let $\hat{\alpha} = \{(j, \hat{\tau}_j) \mid \forall j \in \mathcal{N}\}$ denote the *tentative assignment* in which tuple τ_j is assigned to supplier j . Also, let $\hat{\mathbf{p}}_j^o = (\hat{p}_j^o(\tau) \mid \forall \tau \in \mathcal{T})$ denote the *tentative offer price vector* prescribed by the OEM to supplier j and let $\hat{\mathbf{p}}_j^b = (\hat{p}_j^b(\tau) \mid \forall \tau \in \mathcal{T})$ denote the supplier j 's *tentative bid price vector* in response to $\hat{\mathbf{p}}_j^o$. Of course, in a tentative assignment $\hat{\alpha}$, supplier j is assigned to a non-empty tuple $\hat{\tau}_j$ only if there exists a *tentative agreed price* $\hat{p}_j(\hat{\tau}_j)$ such that $\hat{p}_j(\hat{\tau}_j) = \hat{p}_j^o(\hat{\tau}_j) = \hat{p}_j^b(\hat{\tau}_j)$. Based on the LP2-DLP2 primal-dual pair, $\hat{\alpha}$ represents an LP2 solution while $\hat{\mathbf{p}}_j^o$ and $\hat{\mathbf{p}}_j^b$, for all $j \in \mathcal{N}$, represent a DLP2 solution (See section 4.4).

At the beginning of **OUTSOURCING**, OEM offers $\hat{\mathbf{p}}_j^o$, which is set equal to some *starting price vector*, $\bar{\mathbf{p}} = (\bar{p}(\tau) \mid \forall \tau \in \mathcal{T})$, for every supplier. Given $\hat{\mathbf{p}}_j^o$, supplier j computes each of his tentative bid price $\hat{p}_j^b(\tau)$ in $\hat{\mathbf{p}}_j^b$ through the following price update rule.

$$\hat{p}_j^b(\tau) = \begin{cases} \hat{p}_j^o(\tau), & \text{if } \hat{p}_j^o(\tau) \geq c_j(\tau) \\ \hat{p}_j^o(\tau) + \epsilon, & \text{if } \hat{p}_j^o(\tau) < c_j(\tau) \\ \infty, & \text{if } \bar{p}_j(\tau) < c_j(\tau) \end{cases} \quad (4.38)$$

In practice, if the starting price $\bar{p}_j(\tau)$ is less than supplier j 's cost $c_j(\tau)$, then the supplier can just ignore tuple τ by not submitting bid price on the tuple. Also, in order for the algorithm to terminate in a finite number of iterations, another bidding rule has to be enforced. That is, for any tuple τ , once supplier j sets his bid price $\hat{p}_j^b(\tau)$ equal to $\hat{p}_j^o(\tau) + \epsilon$, he will have to keep the bid price $\hat{p}_j^b(\tau)$ at that value throughout the rest of the algorithm's duration. This prevents the suppliers from changing their minds which may cause the algorithm to run indefinitely.

At any $\hat{\mathbf{p}}_j^b$, supplier j determines the highest profit achievable, U_j^* . The highest profit, however, has to be non-negative, therefore we can write:

$$U_j^* = \max\{0, \max_{\tau \in \mathcal{T}} \hat{p}_j^b(\tau) - c_j(\tau)\} \quad (4.39)$$

OUTSOURCING

```

Set stop = false;  $t = 0$ ;  $\hat{\alpha} = \{\emptyset\}$ ;  $\hat{\mathbf{p}}_j^o = \bar{\mathbf{p}}$ ,  $\mathcal{B}_j^t = \emptyset, \forall j \in \mathcal{N}$ ;
while(stop = false) {
     $t++$ ;
    Each supplier  $j$  updates  $\hat{\mathbf{p}}_j^b$  according to the price update rule shown in (4.38);
    Each supplier  $j$  computes  $\mathcal{B}_j$  through equation (4.40);
    Consumer computes  $\hat{\alpha}$ , by solving TA, the formulation (4.41)–(4.44);
    if( $(\mathcal{B}_j^t = \emptyset, \forall j)$  or  $(\mathcal{B}_j^t = \mathcal{B}_j^{t-1}, \forall j)$  or  $(\hat{\tau}_j \neq \tau(\emptyset), \forall j)$ ) stop = true;
    else Consumer updates  $\hat{\mathbf{p}}_j^o, \forall j \in \mathcal{N}$ , through equation (4.46);
}

```

Figure 4.1 OUTSOURCING algorithm

Based on U_j^* and $\hat{\mathbf{p}}_j^b$, supplier j calculates a *bid set*, \mathcal{B}_j , which reports to the OEM the set of tuple-price pair(s) that maximize his profit. We assume that suppliers are indifferent for the profits that come within ϵ apart. As a result, a bid set \mathcal{B}_j can be defined as:

$$\mathcal{B}_j(\hat{\mathbf{p}}_j^b) = \{(\tau, \hat{p}_j^b(\tau)) \mid \hat{p}_j^b(\tau) - c_j(\tau) + \epsilon > U_j^*\} \quad (4.40)$$

It should be noted that the bid set, \mathcal{B}_j , contains all the tuples that give profit in the range of $[U_j^* - \epsilon, U_j^*]$.

Figure 4.1 illustrates the OUTSOURCING algorithm. At the beginning, all of the suppliers are assigned to empty tuples. The tentative offer price vectors for all suppliers are set equal to the starting price vector. The iteration counter, t , is set to zero, and the bid sets of all the suppliers are set initially to empty sets.

After the initialization stage, we now enter the iterative process. As seen in the **while** loop of Figure 4.1, there are six main steps in each iteration. First, the iteration counter is incremented. Second, each supplier j updates the bid price vector $\hat{\mathbf{p}}_j^b$ through the rule shown in (4.38). This step ensures that each supplier will not incur a loss by accepting price that is lower than his cost. Additionally, if the starting price of a tuple is less than a supplier's

cost, then the supplier will just ignore the tuple by setting a bid price at infinity. Third, with $\hat{\mathbf{p}}_j^b$, each supplier j calculates the bid set, \mathcal{B}_j . Recall that \mathcal{B}_j contains the tuple(s) that maximize supplier j 's surplus (within ϵ accuracy). This \mathcal{B}_j is submitted to the OEM. Fourth, the OEM calculates the tentative assignment, $\hat{\alpha}$, that maximizes his surplus based on the bid sets submitted by the suppliers. This is accomplished by solving the Tentative Assignment model (TA) described by (4.41)–(4.44) as follows.

$$\max \sum_{j \in \mathcal{N}} [v(\tau) - \hat{p}_j^b(\tau)] \cdot z_j(\tau) \quad (4.41)$$

subject to

$$\sum_{\tau \in \mathcal{B}_j} z_j(\tau) \leq 1, \quad \forall j \in \mathcal{N} \quad (4.42)$$

$$\sum_{j \in \mathcal{N}} \sum_{S \ni i} z_j(\tau) \leq 1, \quad \forall i \in \mathcal{I} \quad (4.43)$$

$$z_j(\tau) \in \{0, 1\}, \quad \forall \tau \in \mathcal{B}_j, \forall j \in \mathcal{N} \quad (4.44)$$

We introduce variable $z_j(\tau)$ to indicate whether the tuple (τ) , which is in \mathcal{B}_j , is assigned to supplier j :

$$z_j(\tau) = \begin{cases} 1, & \text{if tuple } (\tau) \text{ is assigned to supplier } j \\ 0, & \text{otherwise} \end{cases} \quad (4.45)$$

The objective (4.41) is to maximize the OEM's surplus. Constraints (4.42) ensure that, in $\hat{\alpha}$, a supplier is assigned to at most one of his favorite tuples. Constraints of type (4.43) restrict that an item can be assigned to at most one supplier. The binary variables are defined by (4.44). Even though TA is an integer program, but because it considers only the tuples in suppliers' bid sets, the number of variables in TA are significantly less than those of IP1.

We can derive the tentative assignment, $\hat{\alpha}$, from TA solution by setting:

$$\hat{\alpha} = \{(j, \hat{\tau}_j) \mid \forall j \in \mathcal{N}\}$$

where,

$$\hat{\tau}_j = \begin{cases} \tau, & \text{if } z_j(\tau) = 1 \\ \tau(\emptyset), & \text{if } \mathcal{B}_j^t = \emptyset \text{ or } \sum_{\tau \in \mathcal{B}_j^t} z_j(\tau) = 0 \end{cases}$$

Fifth, the optimality conditions are verified. These conditions are:

- $\mathcal{B}_j^t = \emptyset, \forall j \in \mathcal{N}$: Given the offer price vector, \mathbf{p}^o , none of the suppliers can find a tuple that generates positive utility.
- $\mathcal{B}_j^t = \mathcal{B}_j^{t-1}, \forall j \in \mathcal{N}$: The bid sets of all the suppliers do not change from the previous iteration.
- $\hat{\tau}_j \neq \tau(\emptyset), \forall j \in \mathcal{N}$: All of the suppliers are assigned to non-empty tuples.

If any one of three conditions is met, the algorithm stops. Sixth, if none of the conditions is met, the OEM then updates the tentative offer price vector, $\hat{\mathbf{p}}_j^o$, for all j in \mathcal{N} that are assigned to empty tuples in that iteration, by setting:

$$\hat{p}_j^o(\tau) = \hat{p}_j^b(\tau) - \epsilon, \quad \forall (\tau) \in \mathcal{B}_j, \forall j : \hat{\tau}_j = \tau(\emptyset) \quad (4.46)$$

Clearly, from iteration t to $t + 1$, if a supplier repeats his bid price for a tuple, then OEM's offer price for that tuple also remains the same.

Following is an example demonstrating the progress of OUTSOURCING.

Example 4 An OEM has demand in the set $\mathcal{I} = \{A, B\}$. . Each of the bundle has to be delivered on a delivery date in $\mathcal{D} = \{Mon, Tue\}$. There are suppliers in the set $\mathcal{N} = \{1, 2\}$ participating the auction. The OEM's valuation vector, \mathbf{v} , and the suppliers' cost vectors, \mathbf{c}_1 and \mathbf{c}_2 , are presented in the following tables.

\mathbf{v}	A	B	AB
Mon	10	8	17
Tue	8	10	15

\mathbf{c}_1	A	B	AB
Mon	5	6	10
Tue	3	5*	8

\mathbf{c}_2	A	B	AB
Mon	4*	7	12
Tue	5	7	10

	(A, Mon)	(B, Mon)	(AB, Mon)	(A, Tue)	(B, Tue)	(AB, Tue)
$\hat{\mathbf{p}}_1^o$	10	8	17	8	10	15
$\hat{\mathbf{p}}_2^o$	10	8	17	8	10	15
$\hat{p}_1^o(\cdot) - c_1(\cdot)$	5	2	7✓	5	5	7✓
$\hat{p}_2^o(\cdot) - c_2(\cdot)$	6✓	1	5✓	3	3	5✓
$v(\cdot) - \hat{p}_1^b(\cdot)$	-	-	0	-	-	0
$v(\cdot) - \hat{p}_2^b(\cdot)$	0	-	0*	-	-	0
$\hat{\mathbf{p}}_1^o$	10	8	15	8	10	13
$\hat{\mathbf{p}}_2^o$	10	8	17	8	10	15
$\hat{p}_1^o(\cdot) - c_1(\cdot)$	5✓	2	5✓	5✓	5✓	5✓
$\hat{p}_2^o(\cdot) - c_2(\cdot)$	6✓	1	5✓	3	3	5✓
$v(\cdot) - \hat{p}_1^b(\cdot)$	0	0	2*	0	0	2
$v(\cdot) - \hat{p}_2^b(\cdot)$	0	0	0	0	0	0
$\hat{\mathbf{p}}_1^o$	10	8	15	8	10	13
$\hat{\mathbf{p}}_2^o$	8	8	15	8	10	13
$\hat{p}_1^o(\cdot) - c_1(\cdot)$	5✓	2	5✓	5✓	5✓	5✓
$\hat{p}_2^o(\cdot) - c_2(\cdot)$	4✓	1	3✓	3✓	3✓	3✓
$v(\cdot) - \hat{p}_1^b(\cdot)$	0	0	2	0	0	2
$v(\cdot) - \hat{p}_2^b(\cdot)$	2*	0	2	0	0	2
$\hat{\mathbf{p}}_1^o$	8	8	13	6	8	11
$\hat{\mathbf{p}}_2^o$	8	8	15	8	10	13
$\hat{p}_1^o(\cdot) - c_1(\cdot)$	3✓	2✓	3✓	3✓	3✓	3✓
$\hat{p}_2^o(\cdot) - c_2(\cdot)$	4✓	1	3✓	3✓	3✓	3✓
$v(\cdot) - \hat{p}_1^b(\cdot)$	2	0	4	2	2*	4
$v(\cdot) - \hat{p}_2^b(\cdot)$	2*	0	2	0	0	2

✓ tuple in bid set

* tuple in tentative assignment

Table 4.1 Progress of the algorithm in Example 2

For convenience, since a bundle has to be delivered on one delivery date, we use the notation $(\mathcal{S}, k_{\mathcal{S}})$ to represent tuples. Also, we write A , B , and AB to indicate bundles $\{A\}$, $\{B\}$, and $\{A, B\}$, respectively. Upon inspection, we can see that the efficient assignment is $\{(1, (B, Tue)), (2, (A, Mon))\}$. The auction progress is shown in Table 4.1.

Consider the first iteration. The starting offer prices in \mathbf{p}_1^o and \mathbf{p}_2^o are set equal to the valuations in \mathbf{v} .¹ Since the offer prices are greater than the suppliers' costs, the suppliers accept the offer prices and make bid prices equal to them. Then, the suppliers compute their profits and determine their bid sets. In the table, the tuples that are contained in the bid sets are marked by “✓” symbol. Based on supplier 1's and supplier 2's bid sets, OEM determines the tentative assignment through TA. In the table, the symbol “-” indicates that the tuple is not in the bid set. An individual supplier-tuple assignment is marked by “*” symbol. The tentative assignment in the first iteration is $\{(1, \tau(\emptyset)), (2, (AB, Mon))\}$. Because supplier 1 is assigned to empty tuple, the offer prices of the tuples in supplier 1's bid set, i.e., tuples (AB, Mon) and (AB, Tue) , decrease by ϵ at the beginning of the second iteration. The process continues until iteration 4, where both supplier 1 and 2 are assigned to non-empty tuples. The final assignment $\hat{\alpha}$ is $\{(1, (B, Tue)), (2, (A, Mon))\}$, while $\hat{\mathbf{p}}_1^o$ is equal to $\hat{\mathbf{p}}_1^b$ and $\hat{\mathbf{p}}_2^o$ is equal to $\hat{\mathbf{p}}_2^b$.

4.6.2 LP Interpretations of the Algorithm's Outputs

In each iteration, OUTSOURCING generates the following outputs: tentative assignment, $\hat{\alpha}$, tentative offer price vectors, $\hat{\mathbf{p}}_j^o$ for all $j \in \mathcal{N}$, and tentative bid price vectors, $\hat{\mathbf{p}}_j^b$, for all $j \in \mathcal{N}$. Through these outputs, we first show in this section that the variables correspond to those of LP2's DLP2's can be derived. These derived variables will be used in proving the algorithm's properties in the next section. Then, we show that the algorithm's outputs constitute LP2's and DLP2's feasible solutions.

¹This is reasonable because we assume that OEM does not have any knowledge about suppliers' costs. Otherwise, by capitalizing on the knowledge about suppliers' costs, the OEM may set lower starting prices and potentially achieve greater surplus. The resulting assignment, however, may not be efficient.

Let $\hat{x}(\cdot)$, $\hat{y}(\cdot)$, $\hat{w}(\cdot)$, $\hat{\pi}_c$, and $\hat{\pi}_j$ be the variables respectively derived from OUTSOURCING that correspond to $x(\cdot)$, $y(\cdot)$, $w(\cdot)$, π_c , and π_j in LP2 and DLP2.

Given the tentative assignment, $\hat{\alpha}$, or equivalently, the TA solution, we define $\hat{x}_j(\tau)$ by letting:

$$\hat{x}_j(\tau) = \begin{cases} z_j(\tau), & \forall \tau \in \mathcal{B}_j^t \\ 0, & \forall \tau \notin \mathcal{B}_j^t \end{cases} \quad (4.47)$$

If $\hat{\alpha}$ is non-empty, then we set $\hat{y}(\hat{\alpha})$ equal to one, otherwise set it to zero.

Given the tentative bid price vector, $\hat{\mathbf{p}}_j^b$, we define $\hat{w}_j(\tau)$ and $\hat{\pi}_c$ by:

$$\hat{w}_j(\tau) = v_j(\tau) - \hat{p}_j^b(\tau) \quad (4.48)$$

and,

$$\hat{\pi}_c = \max_{\alpha \in \mathcal{A}} \sum_{[j,(\tau)] \in \alpha} \hat{w}_j(\tau) \quad (4.49)$$

$$= \max_{\alpha \in \mathcal{A}} \sum_{[j,(\tau)] \in \alpha} v_j(\tau) - \hat{p}_j^b(\tau) \quad (4.50)$$

Lastly, given the tentative offer price vector, $\hat{\mathbf{p}}_j^o$, we define $\hat{\pi}_j$ by:

$$\hat{\pi}_j = \max\{0, \max_{\tau \in \mathcal{T}} \hat{p}_j^o(\tau) - c_j(\tau)\} \quad (4.51)$$

The tentative assignment, $\hat{\alpha}$, is LP2 feasible. This is because the constraints of TA agree with the constraints of LP2 in that they allow each supplier to receive at most one tuple and also they restrict that the bundles assigned to suppliers cannot overlap.

The tentative offer price vectors, $\hat{\mathbf{p}}_j^o$, for all j in \mathcal{N} , and the tentative bid price vector, $\hat{\mathbf{p}}_j^b$, together, represent DLP2 feasible solution defined by constraints (4.23) and (4.24). To see this, for every supplier j and tuple τ , replace $p_j^b(\tau)$ and $p_j^o(\tau)$ with $\hat{p}_j^b(\tau)$ and $\hat{p}_j^o(\tau)$, respectively. Also, replace π_c with $\hat{\pi}_c$. According to the bid price update rule shown in (4.38), it is clear that $\hat{p}^b(\tau) \geq \hat{p}^o(\tau)$ for all τ in \mathcal{T} , which satisfies (4.32). Recall that (4.32) is equivalent to the DLP2 constraints (4.23). Constraints (4.24) of DLP2 are also satisfied because of the definition of $\hat{\pi}_c$ in (4.50).

4.6.3 Properties of the Algorithm

Let V_{LP2}^* be the efficient assignment value of LP2 and V_{OUT} be the assignment value achieved by OUTSOURCING. Since prices are adjusted in a discrete step of ϵ , therefore V_{OUT} can be less than V_{LP2}^* . In such cases, the algorithm causes inefficiency to the economy. In this section, we first show that OUTSOURCING terminates in a finite number of iterations. Then, through the three lemmas which will be proposed, we present a theorem which guarantees the maximum discrepancy between V_{OUT} and V_{LP2}^* , i.e., the inefficiency bound.

Proposition 4.1 OUTSOURCING terminates in a finite number of iterations.

Proof: The algorithm does not terminate in iteration t if, in that iteration, there is at least one tuple for which at least one unassigned supplier agrees (bids) at the tentative offer price. In that case, the tentative offer price of the tuple(s) will decrease by ϵ in iteration $t + 1$. Suppose that the algorithm runs indefinitely, then the agreed price(s) of some tuple(s) must approach $-\infty$. But since the costs to suppliers on all tuples are non-negative, then the agreed prices on any tuple cannot be negative, which contradicts supposition. \square

Lemma 4.4 In any iteration, if $\hat{\tau}_j$ is not equal to $\tau(\emptyset)$, then the following inequalities hold:

$$\hat{p}^o(\hat{\tau}_j) - c_j(\hat{\tau}_j) + 2\epsilon \geq \hat{\pi}_j, \quad \forall j \in \mathcal{N} \quad (4.52)$$

Proof: The complementary slackness condition (4.26) would imply that, for a supplier j and a non-empty tuple (τ) :

$$x_j^*(\tau) = 1 \rightarrow w_j^*(\tau) + \pi_j^* = v_j(\tau) - c_j(\tau) \quad (4.53)$$

Rearranging statement (4.53) we obtain:

$$x_j^*(\tau) = 1 \rightarrow \pi_j^* = p_j^{o*}(\tau) - c_j(\tau) \quad (4.54)$$

where $p_j^{o*}(\tau)$ denotes the final offer price for supplier j on tuple (τ) . The complementary slackness condition (4.26) would imply that supplier j is assigned to tuple (τ) if that tuple maximizes the supplier's profit.

In OUTSOURCING, however, prices are updated in a discrete step, ϵ , therefore complementary slackness condition (4.26) does not always hold. We derive a relaxation of condition (4.26) that implies that in each iteration of OUTSOURCING every supplier receives a non-empty tuple only if it gives profit no less than a certain limit from the maximum profit possible, $\hat{\pi}_j$.

To see this, consider any particular iteration of the algorithm. Each supplier j in \mathcal{N} submits a bid set, \mathcal{B}_j , to the OEM. This bid set \mathcal{B}_j contains all the tuple(s) that give profit within ϵ from the profit-maximizing tuple(s). Based on the suppliers' bid sets, the OEM determines the tentative assignment, $\hat{\alpha} = \{(j, \hat{\tau}_j) \mid \forall j \in \mathcal{N}\}$, through TA that maximizes his surplus. Constraints (4.42) of TA ensure that tuple $\hat{\tau}_j$ in $\hat{\alpha}$ must be chosen from \mathcal{B}_j . According to the definition of \mathcal{B}_j shown in (4.40), for each tuple τ in \mathcal{B}_j , we have:

$$\hat{p}_j^b(\tau) - c_j(\tau) + \epsilon \geq \max\{0, \max_{\tau \in \mathcal{T}} \hat{p}_j^b(\tau) - c_j(\tau)\} \quad (4.55)$$

Due to the suppliers' price update rule, the following relationship holds,

$$\hat{p}_j^o(\tau) \leq \hat{p}_j^b(\tau) \leq \hat{p}_j^o(\tau) + \epsilon \quad (4.56)$$

Further, from (4.39) and (4.50), we have:

$$U_j^* = \max\{0, \max_{\tau \in \mathcal{T}} \hat{p}_j^b(\tau) - c_j(\tau)\} \geq \max\{0, \max_{\tau \in \mathcal{T}} \hat{p}_j^o(\tau) - c_j(\tau)\} = \hat{\pi}_j \quad (4.57)$$

By transforming statement (4.55) using (4.56) and (4.57), we can define the suppliers' profit maximizing condition for OUTSOURCING analogous to (4.54) as:

$$\hat{x}_j(\tau) = 1 \quad \rightarrow \quad \hat{p}_j^o(\tau) - c_j(\tau) + 2\epsilon \geq \hat{\pi}_j \quad (4.58)$$

Inequality (4.58) states that, if a supplier is assigned to a tuple in his bid set, then his profit due to the tuple can be less than $\hat{\pi}_j$, but not less than $\hat{\pi}_j - 2\epsilon$. This condition is satisfied in *every* iteration. Observe that (4.58) is a relaxation of condition (4.54). \square

Lemma 4.5 At termination, if $\hat{\tau}_j$ is equal to $\tau(\emptyset)$, then the following inequalities hold:

$$\hat{p}_j^o(\tau) - c_j(\tau) \leq 0, \quad \forall \tau \in \mathcal{T} \quad (4.59)$$

Proof: At termination, if supplier j is assigned to an empty tuple, then his bid set must be empty, which implies (4.59). Otherwise, if his bid set is not empty, the offer price of a tuple in his bid set will decrease further and the algorithm does not yet terminate. \square

This lemma indicates that, at termination of OUTSOURCING, suppliers who receive no tuples are satisfied with the result because none of the tuples give them positive profit. Note that (4.59) does not necessarily hold in iterations prior to termination. This is due to the fact that in iterations prior to termination, a supplier may be assigned to an empty tuple just because of tie breaking, even though his bid set is not empty.

Lemma 4.6 In any iteration, the following inequality holds:

$$\sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [v_j(\hat{\tau}_j) - \hat{p}_j^o(\hat{\tau}_j)] \geq \hat{\pi}_c \quad (4.60)$$

Proof: First, consider the situation in which an empty assignment is selected. Because i) OEM is a profit-maximizing agent, ii) $v(\tau(\emptyset)) = 0$, and iii) $\hat{p}_j^o(\tau) \leq \hat{p}_j^b(\tau)$, inequality (4.60) always holds under this situation.

Next, we examine the situation in which some non-empty assignment is selected. Recall from (4.50) that:

$$\hat{\pi}_c = \max_{\alpha \in \mathcal{A}} \sum_{(j, \hat{\tau}_j) \in \alpha} v_j(\hat{\tau}_j) - \hat{p}_j^b(\hat{\tau}_j)$$

The following two propositions are essential for the proof of this lemma.

Proposition 4.2 In any iteration, if $\hat{p}_j^b(\tau)$ drops from $\bar{p}_j(\tau)$ to $\bar{p}_j(\tau) - \epsilon$, then supplier j bids for tuple τ .

Proof: In any iteration of OUTSOURCING, given $\hat{\mathbf{p}}_j^o$, each supplier j updates $\hat{\mathbf{p}}_j^b$. According to the OEM's and the supplier's price update rules, a price $\hat{p}_j^b(\tau)$ in $\hat{\mathbf{p}}_j^b$ changes only if supplier j bids for tuple τ in that iteration. \square

Proposition 4.3 Let t' be the iteration that the algorithm terminates. If a supplier bids for a tuple in an iteration $t < t'$, then he will bid for the tuple again in iteration $t + 1$.

Proof: We will show that a tuple that gives maximum profit (within ϵ) to the supplier in iteration t will remain to give maximum profit (within ϵ) in iteration $t + 1$. Let τ' be a particular tuple which is in \mathcal{B}_j^t , and let τ'' be another arbitrary tuple. We denote the utility of supplier j due to a tuple τ in iteration t by $U_j^t(\tau)$. Two possible cases are examined.

Case 1: The tuple τ'' is also in \mathcal{B}_j^t .

Suppose supplier j fails in iteration t . Then the prices of all the tuples in \mathcal{B}_j^t decrease by ϵ in iteration $t + 1$. As a consequence, the utilities on both tuples in iteration $t + 1$ decrease equally by ϵ . The supplier will not bid on tuple τ'' without bidding on τ' .

Case 2: The tuple τ'' is not in \mathcal{B}_j^t .

In this case, we know that $U_j^t(\tau') > U_j^t(\tau'') + \epsilon$. Suppose supplier j fails in iteration t . Then we have: $U_j^{t+1}(\tau') = U_j^t(\tau') - \epsilon$. Obviously, in iteration $t + 1$, $U_j^{t+1}(\tau') > U_j^{t+1}(\tau'')$ and the supplier will not bid on tuple τ'' without bidding on τ' . \square

Propositions 4.2 and 4.3 show that only the prices of the tuples in the bid set have the values less than their starting prices, and the prices of all the tuples that are not in the bid set are still at their starting values, which give zero surplus to the OEM. Therefore, in any iteration, given $\hat{\mathbf{p}}_j^b$, the tentative tuple assignment, $\hat{\alpha}$, determined by TA, is actually the assignment that gives maximum utility possible. As a result, we can write the following equation.

$$\sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} v(\hat{\tau}_j) - \hat{p}_j^b(\hat{\tau}_j) = \max_{\alpha \in \mathcal{A}} \sum_{(j, \tau) \in \alpha} v_j(\tau) - \hat{p}_j^b(\tau) \quad (4.61)$$

Again, due to the OEM's and suppliers' price update rules, for a tuple (τ) , we know that:

$$\hat{p}_j^o(\tau) \leq \hat{p}_j^b(\tau) \leq \hat{p}_j^o(\tau) + \epsilon \quad (4.62)$$

Also, the maximum number of non-empty tuples in an assignment is either the number of suppliers, N , or the number of items, I , whichever is smaller. Therefore, from (4.61) and

(4.62), we have:

$$\begin{aligned} \sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [v(\hat{\tau}_j) - p_j^o(\hat{\tau}_j)] &\geq \sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [v(\hat{\tau}_j) - p_j^b(\hat{\tau}_j)] \\ &= \hat{\pi}_c \end{aligned}$$

which proves the lemma. \square

Inequality (4.60) indicates that, through OUTSOURCING, the OEM selects the bundle assignment that maximizes his surplus. This surplus is equal to his maximum surplus achievable.

Let V_{OUT} be the system's value achieved when OUTSOURCING terminates. Therefore:

$$V_{\text{OUT}} = \sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [v(\hat{\tau}_j) - c_j(\hat{\tau}_j)] \quad (4.63)$$

Also, let V_{LP2}^* and V_{DLP2}^* be the optimal values of LP2 and DLP2, respectively. From Lemma 4.4 to Lemma 4.6, we derive the following theorem.

Theorem 4.3 (Inefficiency Bound) At termination, the value of the assignment through the algorithm, V_{OUT} , satisfies the following:

$$V_{\text{OUT}} + 2 \min\{N, I\}\epsilon \geq V_{DLP2} \geq V_{DLP2}^* = V_{LP2}^* \quad (4.64)$$

Proof: Recall from inequality (4.52) that the inefficiency bound on each supplier is 2ϵ . Summing all of the inequalities over \mathcal{N} , we have:

$$\sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [p_j^o(\hat{\tau}_j) - c_j(\hat{\tau}_j)] + 2 \min\{N, I\}\epsilon \geq \sum_{j \in \mathcal{N}} \hat{\pi}_j \quad (4.65)$$

where $\min\{N, I\}$ is the maximum number of individual supplier-tuple assignments. By adding (4.60) and (4.65) together, we obtain:

$$\sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [v(\hat{\tau}_j) - c_j(\hat{\tau}_j)] + 2 \min\{N, I\}\epsilon \geq \hat{\pi}_c + \sum_{j \in \mathcal{N}} \hat{\pi}_j \quad (4.66)$$

By substituting $\sum_{(j, \hat{\tau}_j) \in \hat{\alpha}} [v(\hat{\tau}_j) - c_j(\hat{\tau}_j)]$ with V_{OUT} , and substituting $\hat{\pi}_c + \sum_{j \in \mathcal{N}} \hat{\pi}_j$ with V_{DLP2} , we finally derive:

$$V_{\text{OUT}} + 2 \min\{N, I\}\epsilon \geq V_{DLP2} \geq V_{DLP2}^* = V_{LP2}^* \quad (4.67)$$

which concludes the theorem. \square

Obviously, as ϵ gets smaller, the inefficiency becomes closer to zero. Furthermore, if ϵ is less than $\frac{1}{\min\{N, I\}}$, then the algorithm always achieves efficient assignments.

4.7 Practical Significance of the Proposed Auction Algorithm

OUTSOURCING, proposed in Section 4.6.1, is a useful tool in today's business. It is a mechanism by which the multi-item multi-attribute trade can be conducted. As shown in Section 4.6.3, the algorithm achieves approximate *efficient* assignment in *competitive equilibrium*. An *efficient* assignment obtained from the algorithm is not only *efficient* in terms of any particular decision factor, such as price, but also in terms of other factors as well. Moreover, the efficient assignment is achieved in competitive equilibrium, which means that every agents, OEM and suppliers, maximizes his utility and is satisfied with the outcome.

Besides the OEM-suppliers setting, we can apply OUTSOURCING to other settings as well. Consider the following simple example. In a logistic setting, a machine manufacturer company A wants to ship a machine from its plant in city C1 to its customer in city C2. Company A considers price and delivery mode, i.e., fast and slow, as the two decision factors. Two transportation companies, X and Y, are bidding for the contract. Each of the transportation companies knows its costs for the fast and slow deliveries. Through OUTSOURCING, the decisions of who delivers the machine, at which delivery mode, and at what price will be determined. The assignment will be efficient and all of the agents, A , X, and Y, will be satisfied with the outcome.

Part III

Supplier's Cost Determination Models and Algorithms

Chapter 5

Literature Survey on Lot-Sizing Problems

5.1 Chapter Overview

In Part II, an auction mechanism for the multi-item multi-attribute outsourcing is developed. We assume that the OEM knows his valuations on all tuples. For suppliers, we require each of them to know his costs of all tuples. In Chapters 6 and 7 of Part III, we introduce the suppliers' cost determination models, represented by the two variants of the lot-sizing problems and develop heuristic algorithms to determine the cost of each tuple.

In this chapter, the literature related to our research will be reviewed. The chapter is divided into four main sections. Section 5.2 presents an overview of the Production and Inventory Planning Problem (PIPP), specifically the problem involving the Dynamic Lot-Sizing Model (DLSM). It describes some relevant papers that had major impact on the PIPP research community. Also, this section proposes model attributes that can be used to classify the DLSM. Section 5.3 and 5.4 focus on the literature related to classes of the DLSM for simple product structures, namely Single-Level Single-Item Models and Single-Level Multi-Item Models respectively. Section 5.5 deals with models handling complex product structures, called Multi-Level Models. Lastly, Section 5.6 concerns itself with the literature revolving around the two important concepts, the planning horizon and the forecast horizon. Since our research considers the deterministic environment, the stochastic PIPP is not reviewed in this chapter.

5.2 Dynamic Lot-Sizing Problems

The Production and Inventory Planning Problem (PIPP), in its broadest sense, addresses the supplier's decision problems that involve the facility planning and the use of his limited resources to fulfill his current or prospect OEMs' demands in the most efficient fashion. From this definition, it is clear that PIPP incorporates a wide variety of decision processes, ranging from a long-term decision process (e.g., decisions on facility acquisition) to a medium-term decision process (e.g., monthly or weekly decisions on work-force allocation and production levels) and finally to a short-term decision process (e.g., daily decisions on work-force and production scheduling). In this section, however, we will focus our attention only on the medium-term decision process in which time horizon is divided into multiple discrete periods. More specifically, we will consider only the literature that addresses the questions of what, when, and how much to produce in order to fulfill OEMs' demands, and, at the same time, minimize relevant costs. However, the literature along this line of research is so enormous that we will not attempt to make an exhaustive review. Instead, we will point to major developments that have happened in this research area. Some extensive reviews can be found in Bahl et al. [6], Hackman and Leachman [42], Potts and van Wassenhove [82], and Kuik and Salomon [55].

Wagner and Whitin [97] originated the Dynamic Lot Size Model (DLSM), in which the production level of each and every time period has to be determined such that the OEMs' deterministic demands in each and every time period are met with minimal cost. Their model has attracted a lot of interest from research community. Most of the subsequent PIPP models have been developed based on this original DLSM. It is worthwhile to note that the *dynamic* term used in DLSM refers to the varied demand rates over time periods, in contrast with the constant demand rate case previously introduced by Harris [43].

The original DLSM proposed by Wagner and Whitin [97] considers a system in which an uncapacitated facility has to make production level decision x_t responding to the deterministic demand d_t for all period $t \in \Omega = \{1, 2, \dots, T\}$ in order to minimize total cost:

$$\min \sum_{t=1}^T (s_t \delta(x_t) + h_t I_t) \quad (5.1)$$

where s_t and h_t are the setup cost and the per unit per period inventory holding cost, respectively. $\delta(x_t) = 1$ if $x_t > 0$, and $\delta(x_t) = 0$ otherwise. The inventory at the end of period t is denoted by I_t , while I_0 denotes the beginning inventory of period 1. In this model, there are two sets of constraints. The first one is the inventory balance constraints:

$$I_{t-1} + x_t - d_t = I_t, \quad \forall t \in \Omega \quad (5.2)$$

where the inventory at the end of period t must be equal to the beginning inventory of that period plus the production quantity minus the demand in that period. The second set of constraints is non-negativity:

$$x_t \geq 0, \quad \forall t \in \Omega \quad (5.3)$$

$$I_t \geq 0, \quad \forall t \in \Omega \quad (5.4)$$

For convenience purpose, we will refer to the model presented in (5.1) through (5.4) as the *Wagner-Whitin* model.

The basic trade-off that one has to consider here is the balance of the economy of scales motive and the inventory cost motive. For example, if we produce a lot at the current period to cover some of the future demand, we can probably save some setup cost for the future period. But in so doing, the inventory cost is automatically increased.

It is found in Wagner and Whitin [97] that, there exists an optimal program such that $I_{t-1} x_t = 0$. In other words, there will be a production in period t if and only if the beginning inventory of that period is zero. However, this condition holds for this model because it assumes constant production cost over all time periods.

The most important contribution of Wagner and Whitin [97] is probably the *Planning Horizon Theorem* which identifies the necessary condition to be met in order to divide the

original T -period problem into smaller sub-problems that can be considered separately. A period t is called *planning horizon*, if the optimal production levels for the sub-problem containing period 1 through t remain optimal in the problem with greater length. This concept is very useful in that we can find the optimal production levels for some of the near future periods without having the complete information of the far future periods. With this powerful property, Wagner and Within develop an $O(T^2)$ forward algorithm that solves the problem.

A wide variety of research has since adopted and further developed the *planning horizon* concept, including the research involving the determination of the *forecast horizon*. The two concepts will be discussed in more detail in section 5.6.

As we can see, the *Wagner-Whitin* model does not completely represent realistic manufacturing systems. For example, it assumes unlimited capacity, constant production cost, and single-item production system. Hence, during the past decades, there have been many efforts in extending the model and improving the solution methods so that the DLSP is more practical.

Before we investigate in more detail, it is certainly helpful to be able to classify the DLSP-related research. This could be done by clearly identifying DLSP-related research attributes. Followings are the research attributes that are deemed relevant.

Number of products With regard to this attribute, papers can be divided into two main groups. The first group deals with systems that produce a single product, or so-called *single-item* systems. The second group deals with systems that produce multiple products, or so-called *multi-item* systems.

Product structure Different DLSP-related research could study models considering different kinds of product structures. A work flow network of input-transformation-output nodes and output-input arcs can be useful in representing a product structure. A product structure is called *single-level* when all input materials are supplied exogenously to the model. A single node, in this case, represents the work flow network. On the other hand, a

product structure is said to be *multi-level* if the work flow network consists of at least a pair of nodes connected by an output-input arc.

Capacity limitation With regard to this attribute, clearly, papers can be divided into two main groups: one assumes infinite production capacity while the other assumes finite production capacity.

Objective function Most DLSSM-related research has a common objective of minimizing some total cost. In the cost function, however, differences in cost components still exist amongst different models. For example, some papers allow backlogging while some do not.

Property of decision variables The complexity of the problem very much depends on the decision variables. If they are restricted to be integer, for example, some problems could become very hard to solve for the exact solutions.

Solution methods Even with the same problem setting, different researchers could solve the problem using different solution methods. Equally true is that one solution method for a problem setting could prove to be useful for solving some other problems with different settings as well.

With all these attributes in mind, we classify the DLSSM-related research into three streams of research—*Single-Level Single-Item Models*, *Single-Level Multi-Item Models*, and *Multi-Level Models*.

5.3 Single-Level Single-Item Models

Federgruen and Tzur [31], Wagelmans et al. [96], and Aggarwal and Park [3] consider a generalized version of the *Wagner-Whitin* model such that, instead of constant production cost for over all periods, the production cost is a fixed-plus-linear function. Consequently,

instead of (5.1), the objective function now becomes:

$$\min \sum_{t=1}^T (s_t \delta(x_t) + p_t x_t + h_t I_t) \quad (5.5)$$

where p_t is unit production cost in period t . Independently, they develop algorithms that run in $O(T \log T)$ time. Federgruen and Tzur [31] also introduce two rationales that prevail for maintaining inventories in systems. One is the *cycle stock motive* by which economies of scale motivates the over-production in the current period, while the excess is stored for future periods. The second is the *speculative motive*, in which case, increasing demand is expected and motivates the over-production in the current period. They present $O(n)$ -time and space algorithm for two special cases—models *without* speculative motives for carrying stock and models with nondecreasing setup costs.

Zangwill [101] is the first to allow backlogging in the DLS model. Unlike (5.1), he considers piecewise concave cost functions:

$$\min \sum_{t=1}^T (P(x_t) + H_t(I_t^+) + \pi_t(I_t^-)) \quad (5.6)$$

where $P(x_t)$ is a concave production cost function, $H_t(I_t^+)$ is a concave holding cost function in period t on the interval $[0, +\infty)$, and $\pi_t(I_t^-)$ is a concave shortage cost function in period t on the interval $(-\infty, 0]$. Obviously, this model does not include constraints (5.4). In Zangwill [101], a dynamic programming algorithm is employed to calculate the minimum cost schedule. The DLS model with backlogging is also studied in many research papers, including Blackburn and Kunreuther [20] and Morton [74]. In Blackburn and Kunreuther [20], the authors develop planning horizons for the cases where the marginal production cost is either constant or varied. A forward algorithm which is a generalization of Wagner and Whitin [97] is also presented. Considering the special case in which costs are linear and stationary over time, Morton [74] develops planning horizon procedures and also presents an improved algorithm for the problem that is comparable in difficulty to the algorithm for the *Wagner-Whitin* model presented in Wagner and Whitin [97].

Capacitated DLS models also attract a great deal of attention from researchers. With constraints (5.2) through (5.4), Florian and Klien [37] studies DLS model with concave production costs and concave holding costs:

$$\min \sum_{t=1}^T (P(x_t) + H_t(I)) \quad (5.7)$$

and the capacity constraint:

$$x_t \leq C, \quad \forall t \in \Omega \quad (5.8)$$

where C is a constant capacity limit for all periods. They characterize the structure of an optimal solution and it is then used in a dynamic programming algorithm. Their algorithm runs in $O(T^4)$ time. The same setting but with backlogging (i.e., with (5.6) as an objective function and without constraints (5.4)) is also studied in the same paper. Lotfi and Yoon [61] later extend the non-backlog model of Florian and Klien [37] such that capacity can vary over periods by replacing (5.8) with

$$x_t \leq C_t, \quad \forall t \in \Omega \quad (5.9)$$

where C_t is the capacity limit of period t . Optimal properties, in addition to those presented in Florian and Klien [37], and an exact solution method are developed. For the case in which production costs are concave but the holding costs are linear, and backlogging is not allowed, van Hoesel and Wagelmans [92] develop an $O(T^3)$ time algorithm. Their algorithm consists of two phases, first finding the optimal solutions for the subplans $(t_1, t_2), 1 \leq t_1 \leq t_2 \leq T$, then determining an optimal solution of the overall problem. A model with cost structure consists of piecewise linear production costs and general holding costs is investigated by Shaw and Wagelmans [88]. They present a dynamic programming procedure that runs in $O(T^2 \bar{q} \bar{d})$, where \bar{q} and \bar{d} are average values required to represent the production cost functions and average demand, respectively.

Hindi [44] studies a single-item, capacitated, lot-sizing model with *start-up* and *reservation costs*, which is first described by Karmarkar et al. [48]. The model does not allow backlogging. A start-up cost in period t , λ_t , is incurred in period t if a production facility

is switched from 'off' in period $t - 1$ to 'on' in period t . A reservation cost in period t , γ_t , however, is incurred in any period that a production facility is 'on'. It is interesting to note that, in some situations, it might be profitable to turn the facility on without producing any products, i.e., $y_t = 1$ but $x_t = 0$. This could happen when, for example, the start-up cost λ_t is high compared with the reservation cost γ_t . With regard to the mathematical model of this problem, the objective function can be written as:

$$\min \sum_{t=1}^T (\lambda_t w_t + \gamma_t y_t + p_t x_t + h_t I_t) \quad (5.10)$$

where,

$$w_t = \begin{cases} 1 & \text{if the production facility was 'off' in period } t - 1 \text{ and 'on' in period } t, \\ 0 & \text{otherwise} \end{cases}$$

$$y_t = \begin{cases} 1 & \text{if the production facility is 'on' in period } t, \\ 0 & \text{otherwise} \end{cases}$$

With the presence of setups the capacity constraints now become

$$x_t \leq C_t y_t, \quad \forall t \in \Omega \quad (5.11)$$

In addition to constraints (5.2) through (5.4) and (5.11), the following set of constraints is included in the model,

$$y_t - y_{t-1} \leq w_t, \quad \forall t \in \Omega \quad (5.12)$$

where it ensures the correct activation of w_t . Solving this problem with standard branch-and-bound procedure typically requires a tremendous computational effort, due to large gap between the optimal solution value of the LP-relaxation problem and that of the original. An algorithm based on an alternative model that is tighter is proposed.

Apart from the models presented above, there are several DLS variants that are interesting, especially from the practical point of view. It should be noted that all of the DLS

models that have been discussed assume *rigid* demand delivery time (i.e., a supplier must finish the requested products *on-time*, otherwise either inventory holding cost or shortage cost is incurred). Lee et al. [58] suggest that under typical supply contracts, the OEM offers a grace period, which they call *demand time window*, during which a particular demand can be satisfied with no penalty. They denote the time window by $[E_t, L_t]$ for demand d_t , where $E_t \leq L_t$. With this motivation, they study two specific single-item DLS models. The first one does not allow backlogging. In this context, it means that a demand d_t must be delivered during its corresponding time window $[E_t, L_t]$. The second model allows backlogging to occur. By that, it means that d_t could not be delivered earlier than E_t , but can be delivered later than L_t with shortage penalty. An $O(T^2)$ algorithm is proposed for the first model, while an $O(T^3)$ is proposed for the second model.

All of the aforementioned models assume that the products under consideration are *age-independent*, i.e., their quality and quantity do not degrade over the stocking periods. This is certainly not a realistic assumption for perishable products. Nahmias [76] discusses in detail about the issue of perishability. Perishable inventory is classified into fixed and variable lifetime inventories. For the first type, the inventory becomes obsolete after a fixed amount of stocking periods. The second perishable inventory type represents inventory whose value is decreasing according to the length of stocking periods. A DLS model for perishable inventory where stock deterioration rates depend on both the stocks' ages and their periods of production is studied by Hsu [46]. It is shown that the model is equivalent to a minimum cost network flow problem on a specially constructed network with flow loss. Some structural properties of the optimal solutions are given and a dynamic programming algorithm that solves the model in polynomial time is developed.

5.4 Single-Level Multi-Item Models

Let $\mathcal{G} = \{1, 2, \dots, G\}$ be the set of products that the production facility can produce. In general, it is assumed that in each time period, the production facility is allowed to produce more than one product. Coupled with limited capacity, the problem is called the

capacitated lot-sizing problem (CLSP) or sometimes simply the *large bucket problem* (See Eppen and Martin [30]). This type of model is suitable for the development of medium-term production planning in which the time horizon is coarsely divided into rather large time periods. However, under the circumstances that the time horizon is divided into very small pieces and short-term production planning is being developed, it might be reasonable to assume that the facility will produce only one product (one lot) in each time period. If it is assumed further that the production level for each period could be either at full capacity or zero, the problem is called the *discrete lot-sizing and scheduling problem* (DLSP) (Fleischmann [34]), or simply *small bucket problem* (Eppen and Martin [30]). As the name suggests, DLSP involves itself with sequencing decisions of the lots, while CLSP does not involve sequencing decisions of the lots within a period.

Focusing first on the large bucket problem. The mixed-integer programming model of the problem can be formulated as follows.

$$\text{(CLSP)} \quad \min \sum_{g=1}^G \sum_{t=1}^T (s_g y_{gt} + h_g I_{gt}) \quad (5.13)$$

subject to

$$I_{g(t-1)} + x_{gt} - d_{gt} = I_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.14)$$

$$p_g x_{gt} \leq C_t y_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.15)$$

$$\sum_{g=1}^G p_g x_{gt} \leq C_t \quad \forall t \in \Omega \quad (5.16)$$

$$y_{gt} \in \{0, 1\} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.17)$$

$$I_{gt}, x_{gt} \geq 0 \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.18)$$

The objective function (5.13) is to minimize the total sum of setup costs and holding costs. Equation (5.14) represents the set of inventory balance constraints. Constraints (5.15) properly activate the set state binary variable for product j in period t , y_{gt} . From (5.15), there will be exactly one setup for each item produced in a period, i.e., a lot will not continue over the end of a period. By constraints (5.16), given processing time p_g for

all j in \mathcal{G} , the sum of production levels of all products in any period t in Ω is restricted to be within the capacity limit C_t . (5.16). The setup variable y_{gt} is restricted to be binary by 5.17. Lastly, 5.18) forces inventory level variables I_{gt} and production level x_{gt} to take only non-negative values.

Florian et al.[36] and Bitran and Yannasse [19] show that to optimally solve CLSP is \mathcal{NP} -hard. In fact, a two-item model could prove to be \mathcal{NP} -hard, even though its single-item version is considered *easy* (See Bitran and Yannasse [19]). Moreover, if setup times are incorporated, determining a feasible solution is \mathcal{NP} -complete (Maes et al. [63]). Despite its prohibitive complexity, enormous research effort has focused on developing exact algorithms (e.g., Constantino [27], Eppen and Marin [30], and Thizy and van Wassenhove [90]) as well as approximate algorithms (e.g., Hindi [45] and Walser et al. [98]). A survey and computational comparison of the heuristics developed for multi-item models can be found in Maes and van Wassenhove [65] and Maes and van Wassenhove [64], respectively.

We now consider DLSP. It can be formulated as a mixed-integer programming model as follows.

$$\text{(DLSP)} \quad \min \sum_{g=1}^G \sum_{t=1}^T (s_g w_{gt} + h_g I_{gt}) \quad (5.19)$$

subject to

$$I_{g(t-1)} + x_{gt} - d_{gt} = I_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.20)$$

$$p_g x_{gt} = C_t y_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.21)$$

$$\sum_{g=1}^G y_{gt} \leq 1 \quad \forall t \in \Omega \quad (5.22)$$

$$y_{gt} - y_{g(t-1)} \leq w_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.23)$$

$$y_{gt} \in \{0, 1\} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.24)$$

$$I_{gt}, x_{gt}, w_{gt} \geq 0 \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.25)$$

Superficially, the MIP model of DLSP looks similar to that of CLSP. The objective function (5.19), like in the CLSP case, is still to minimize the total sum of setup costs and

holding costs. Also, equation (5.20) is a set of inventory balance constraints, like equation (5.14) of the CLSP. However, there are some significant differences between the two models. DLSP assumes 'all or nothing' production, i.e., in each period, if a product is produced, it will take up all the capacity available to it, otherwise, that product will not be produced at all. Equation (5.21) takes care of this assumption. To ensure that in each period, there will be at most one product produced, constraints (5.22) are imposed. For DLSP, it is also assumed that setups occur in period t if and only if there is production of product j in that period, but not in period $t - 1$. In other words, variable w_{gt} will be activated if and only if y_{gt} is 1 and $y_{g(t-1)}$ is 0. The variable w_{gt} is introduced in DLSP to indicate the setup state of product j in period t . Constraints (5.24) and (5.25) are the binary condition for y_{gt} and the nonnegativity constraints for I_{gt} , x_{gt} , and w_{gt} respectively. Interestingly enough, there is no need to impose the binary condition on variable w_{gt} , due to the effects of constraints (5.23) and (5.24).

DLSP is known to be \mathcal{NP} -hard. Solving for a feasible solution, however, can be done in polynomial time. Important papers dealing with DLSP and its variants include Fleischmann [34], Fleischmann [35], Magnanti and Vachani [67], Cattrysse et al. [22], and Salomon et al. [86]. The reader is referred to Salomon [85] for more details regarding computational complexity and additional description of DLSP-related models.

The solution methods developed for the single-level multi-item model can be roughly classified into four major approaches, which are presented as follows.

Relaxation-Based Approaches A relaxation of an integer programming problem expands the solution space without discarding the original solutions. Hence, it can only improve the objective. For minimization problems, as considered here, a relaxed solution identifies the lower bound of the integer optimal solution. Apart from the bounding purpose, a relaxed solution is sometimes used as an initial point in a heuristic that finds a feasible and near optimal solution. Two primary relaxation schemes are linear programming relaxation, or *LP*

relaxation, and *lagrangean relaxation*. Since the majority of the research papers in single-level multi-item model employ the lagrangean relaxation as part of their solution methods, a brief overview of the relaxation scheme is presented below.

Consider the following *minimization* integer program

$$z_{IP} = \min\{\mathbf{c}\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathcal{Z}_+^n\} \quad (5.26)$$

where \mathcal{Z}_+^n is the set of nonnegative integral n -dimensional vectors, \mathbf{c} is n -dimensional data vectors, variables $\mathbf{x} = \{x_1, \dots, x_n\}$, and \mathbf{A} is an $m \times n$ data matrix. Suppose that the constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ in the IP can be partitioned into two sets of constraints $\mathbf{A}_1\mathbf{x} \leq \mathbf{b}_1$ and $\mathbf{A}_2\mathbf{x} \leq \mathbf{b}_2$, representing *easy* and *complicating* sets of constraints, respectively. Also, \mathbf{A}_1 has $m_1 < m$. The Lagrangean relaxation of the IP with respect to constraints $\mathbf{A}_1\mathbf{x} \leq \mathbf{b}_1$ can be written as

$$z_{LR}(\mathbf{u}) = \min\{\mathbf{c}\mathbf{x} + \mathbf{u}(\mathbf{A}_1\mathbf{x} - \mathbf{b}_1) : \mathbf{x} \in \mathcal{Q}\} \quad (5.27)$$

where $\mathbf{u} \in \mathcal{R}_+^{m_1}$ and $\mathcal{Q} = \{\mathbf{x} \in \mathcal{Z}_+^n : \mathbf{A}_2\mathbf{x} \leq \mathbf{b}_2\}$. From (5.26) and (5.27), constraints $\mathbf{A}_1\mathbf{x} \leq \mathbf{b}_1$ are said to be *dualized*, and the m -dimensional row vector \mathbf{u} contains *dual prices*, one for each constraint. It is desirable to maximize $z_{LR}(\mathbf{u})$ so that the tightest lower bound could be obtained. The resulting maximization problem is called the *langrangean dual*, which can be formally defined as

$$z_{LD} = \max_{\mathbf{u}}\{z_{LR}(\mathbf{u})\} \quad (5.28)$$

There are two prominent iterative methods that are used for solving the langrangean dual problem (5.28), namely, *column generation* and *subgradient optimization*. The reader is referred to Magee and Glover [66] and Nemhauser and Wolsey [77] for more detail about Langrangean relaxation technique.

Numerous solution procedures based on the relaxation of the original MIP model have been developed. Most of them use, in particular, the Langrangean relaxation. Thizy and van Wassenhove [90] study CLSP and develop a heuristic procedure based on a Langrangean relaxation of the capacity constraints (5.16). By solving a transportation problem using

solutions determined by Dixon and Silver [29]’s algorithm, the initial value of the Lagrangean multipliers is obtained. The Lagrangean multipliers are updated by the subgradient optimization method. Examining the same problem as Thizy and van Wassenhove [90], Trigeiro [91] develops a *dual-cost heuristic* (DCH). The DCH algorithm also relaxes the capacity constraints (5.16) and, hence, the resulting set of uncapacitated single-item problems can be easily solved by dynamic programming, as in Wagner and Whitin [97]. Since the relaxed solutions could be infeasible, a separate heuristic, called *DCH smoothing routine*, adjusts the relaxed solutions from which a feasible solution is derived. Other research efforts using Lagrangean-based solution methods include Lozano et al. [62] and Millar and Yang [73].

For DLSP, Fleischmann [34] develop a branch-and-bound procedure using Lagrangean relaxation for lower bound determination purpose. The problem therefore is decomposed into uncapacitated single-item problems, as in the CLSP case. However, Wagner and Whitin [97]’s algorithm is not applicable to the relaxed DLSP, so a special dynamic program is developed. Regarding the branching process, a product is assigned to be produced in a certain period, starting from the last period and proceeding backwards to the first period. Other efforts using Lagrangean-based approaches for DLSP include Fleischmann [35] and Cattrysse et al. [22]

Variable Redefinition Approaches It is typical that the optimal solution value of a minimization MIP problem is significantly larger than its LP-relaxation’s counterpart. Hence, a straightforward branch-and-bound approach, that employs the LP-relaxation procedure for lower bound determination purpose, would naturally take enormous solution time. *Variable Redefinition* is an alternative approach that, when implemented, could give tighter lower bounds.

Let $\text{conv}(\mathcal{S})$ be the convex hull of the solution set \mathcal{S} and $\bar{\mathcal{P}}$ be the linear programming relaxation of problem \mathcal{P} . \mathcal{X} denotes the feasible region of the problem and \mathcal{Z} denotes the

feasible region in the new variable space in which $L(\text{conv}(\mathcal{Z})) = \text{conv}(\mathcal{X})$, where L is a linear transformation.

Suppose there is an original MIP model, called \mathcal{P}_1 . Often, by dropping the *complicating* constraints in an \mathcal{P}_1 , a *special structure subproblem* emerges. The basic idea behind the variable redefinition approach is to reformulate this special structure subproblem, which could either consist of an entirely new set of variables, or a subset of the existing variables plus some new *auxiliary* variables. Based on these new variables, \mathcal{P}_1 with feasible region \mathcal{X} can be reformulated as another MIP model, called \mathcal{P}_2 , with feasible region \mathcal{Z} . \mathcal{X} and \mathcal{Z} must be *equivalent*, i.e., $L(\text{conv}(\mathcal{Z})) = \text{conv}(\mathcal{X})$. The reformulation, in many cases, gives \mathcal{P}_2 model such that the feasible region of $\bar{\mathcal{P}}_2$ is either equal to $\text{conv}(\mathcal{Z})$ or is a close approximation of it. Consequently, by solving $\bar{\mathcal{P}}_2$, either the optimal solution to \mathcal{P}_2 (from which the optimal solution to \mathcal{P}_1 could be derived) or a supposedly tight lower bound can be obtained. Martin [68] thoroughly elaborates the theory of Variable Redefinition.

In Eppen and Martin[30], the variable redefinition is applied to the multi-item capacitated lot-sizing problems. They reformulate a single-item model studied by Karmarkar et al. [48] and present the fundamental concepts of variable redefinition. A three-step procedure for implementing the variable redefinition process is proposed. First, develop an alternative formulation \mathcal{Z} such that $\text{conv}(\mathcal{Z}) = \bar{\mathcal{Z}}$, where $\bar{\mathcal{Z}}$ is a set defined from \mathcal{Z} by dropping the integrality constraints. Second, find a linear transformation L from \mathcal{Z} to \mathcal{X} such that \mathcal{Z} is equivalent to \mathcal{X} under L . Third, select an appropriate model from two prototype models. The multi-item lot-sizing problems of both small bucket type (Lasdon and Terjung [57]) and large bucket type are also reformulated. Their results show that the LP-relaxation of the reformulated models produce bounds equal to those of Lagrangean relaxation or column generation methods. Moreover, the approach is effective especially when $G \gg T$. Thus, an important benefit of their approach is that special programming requirements of the Lagrangean relaxation or column generation algorithms can be avoided and *off-the-shelf* codes could then be used to solve an MIP problem.

Cutting Plane Approaches A multi-item model is usually formulated as an MIP. An MIP, in turn, is usually solved by the branch-and-bound algorithm. However, using the branch-and-bound algorithm alone could result in a very slow solving process. For this reason, the cutting plane approach is sometimes employed (either by itself, or together with branch-and-bound) in hope that it can speed up the solving process.

A cutting plane algorithm developed by Gomory [39] is the first provably finite cutting plane algorithm proposed for solving integer programs (Magee and Glover [66]). The basic idea behind the cutting plane algorithm is that a set of constraints, called cuts, is added to the current LP until the optimal basic feasible solution takes on integer values. The cuts, in effect, reduce the LP feasible region, without excluding any feasible integer solutions. Also, once added to the current LP, they make the current fractional solution infeasible. Cuts can be determined either by using the information from linear programming tableau (Gomory [39]) or exploiting the special structure of the problem under consideration. While the first approach is a general method that can be used in any IP, it can be very slow. The second approach, on the other hand, has to be customized for a specific problem, but it could be very efficient. In recent years, a number of researchers have turned their attention to the second approach. That results in the proliferation of research employing cutting plane approaches. For more details about the cutting plane approaches, the reader is referred to Taha [89], Salkin and Mathur [84], and Ignizio and Cavalier [47].

Constantino [27] studies the problem which is known as the Deterministic Dynamic Product Cycling Problem (Karmarkar and Shrage [49]). The problem requires that only one product be processed in a period. Capacity constraints are also imposed. The problem is formulated as an MIP. Through the study of the polyhedral structure of an MIP formulation of the single-item version, the author derives valid inequalities for the multi-item applications that are used in the cutting plane algorithm. In the experiments, a cutting plane plus branch-and-bound algorithm is performed. Cuts are generated only in the first node of the branch-and-bound tree. Problems with five items and up to 36 periods are solved to optimality. Also observed is that, besides the problem size, the factors determining problem

difficulty are probably the value of the capacity and the demand pattern. The same approach employed in other multi-item models can be found in Barany [8], Leung et al. [60], Pochet and Wolsey [80], and Magnanti and Vachani [67]. A survey on this line of research can be found in Pochet [81].

5.5 Multi-Level Models

A product structure can be represented as a work-flow network, in which a node represents a transformation process and an arc represents the input-output direction. A structure is called *single-level* if the work-flow network consists of a single node. On the other hand, a structure is said to be *multi-level* if there are at least a pair of nodes connected by an input-output arc. Multi-level models can be classified based on the type of the network (See Kuik et al. [55]): *Serial Structure*: a (connected) work-flow network in which each node has at most one incoming arc and one outgoing arc, *Assembly Structure*: a (connected) network in which each node has at most one outgoing arc, *Arborescent Structure*: a (connected) network in which each node has at most one incoming arc, and *General Structure*: a network that does not fit into any of the above classifications.

Due to the fact that the multi-level production naturally causes internal demands (i.e., requirement for components) to be dependent in terms of a predecessor-successor relationship, multi-level lot-sizing models are very hard to solve. However, tremendous attention has been given to them. The reader is referred to Bahl et al. [6] and Kuik et al. [55] for more detailed reviews on this research direction. Some important research efforts will be discussed here.

Without capacity constraints, the multi-level lot-sizing problem is studied by Afentakis et al. [2]. They convert the general structure problem into a simple assembly structure with additional constraints and solve the transformed problem by branch-and-bound based procedure in which a Lagrangean relaxation method is embedded. The computational complexity of the uncapacitated multi-level lot-sizing problems can be found in Arkin et al. [5].

With capacity constraints, two important multi-level lot-sizing problems, namely the multi-level capacitated lot-sizing problem (MLCLSP) and a multi-level discrete lot-sizing and scheduling problem (MLDLSP), will be discussed. The problems are described as mathematical models because they provide exact problem descriptions. For understanding purposes, the mathematical models are for the single-machine case only.

Not surprisingly, the MLCLSP is directly extended from the CLSP discussed in the single-level lot-sizing section. A basic version of MLCLSP is formulated as an MIP as shown below, where $\mathcal{K}(l)$ is a set of successors (i.e., products that need product l as component) and a_{lg} is a *gozinto* factor (i.e., units of product l needed for one unit of product g).

$$\text{(MLCLSP)} \quad \min \sum_{g=1}^G \sum_{t=1}^T (s_g y_{gt} + h_g I_{gt}) \quad (5.29)$$

subject to

$$I_{g(t-1)} + x_{gt} - d_{gt} - \sum_{g \in \mathcal{K}(l)} a_{lg} x_{gt} = I_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.30)$$

$$p_g x_{gt} \leq C_t y_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.31)$$

$$\sum_{g=1}^G p_g x_{gt} \leq C_t \quad \forall t \in \Omega \quad (5.32)$$

$$y_{gt} \in \{0, 1\} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.33)$$

$$I_{gt}, x_{gt} \geq 0 \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.34)$$

$$I_{j0} = I_{gt} = 0 \quad \forall g \in \mathcal{G} \quad (5.35)$$

Apparently, MLCLSP looks very much like CLSP, except for the inventory balance constraints (5.30), in which an additional term $(-\sum_{g \in \mathcal{K}(l)} a_{gl} x_{gt})$, representing internal demand for product l , is added. Other than that, the objective (5.29) is to minimize total setup and inventory costs. Inequality (5.31) activates setup variable y_{gt} when positive production occurs. Capacity constraints is imposed by inequality (5.32). Constraints (5.33) to (5.35) properly restrict conditions of the decision variables y_{gt} , x_{gt} , and I_{gt} .

With minor modification, MLDLSP can be directly derived from DLSP. As is the case for MLCLSP, MLDLSP is different from DLSP only in the inventory balance constraints

(5.37), in which the term $(-\sum_{g \in \mathcal{K}(l)} a_{gl}x_{gt})$ is added (See equation (5.20) for comparison).

We now present the MLDLSP.

$$\text{(MLDLSP)} \quad \min \sum_{g=1}^G \sum_{t=1}^T (s_g w_{gt} + h_g I_{gt}) \quad (5.36)$$

subject to

$$I_{g(t-1)} + x_{gt} - d_{gt} - \sum_{g \in \mathcal{K}(l)} a_{gl}x_{gt} = I_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.37)$$

$$p_g x_{gt} = C_t y_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.38)$$

$$\sum_{g=1}^G y_{gt} \leq 1 \quad \forall t \in \Omega \quad (5.39)$$

$$y_{gt} - y_{g(t-1)} \leq w_{gt} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.40)$$

$$y_{gt} \in \{0, 1\} \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.41)$$

$$I_{gt}, x_{gt}, w_{gt} \geq 0 \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (5.42)$$

The *all or nothing production* condition is still preserved by equation (5.38). Constraints (5.39) stipulate that there is at most one product being produced in each period. According to constraints (5.40), a setup cost is incurred on period t only if there is a production of product i in period t but the product was not set up in period $t - 1$. Constraints (5.41) and (5.42) enforce the proper conditions on the decision variables y_{gt} , I_{gt} , x_{gt} , and w_{gt} .

Billington et al. [18] review and further investigate mathematical programming approaches to capacitated multi-level production systems. A very general IP model for the MLCLSP-type problem that considers overtime, undertime, setup time, and multi-machine environment, is proposed. The authors suggest the use of *Product Structure Compression* to reduce the size of the problem while retaining necessary information in complete detail. For a general product structure and a single bottleneck facility, Billington et al. [17] propose a branch-and-bound algorithm that uses Lagrangean relaxations to generate lower bounds. They report a rapid growth of computational time as a function of problem size. Kuik et al. [56] investigate the performance of heuristic procedures based on linear programming, simulated

annealing, and tabu search for the multi-level lot-sizing problems in bottleneck assembly systems. According to their experimental results, simulated annealing and tabu search perform well compared to the pure LP-based heuristic. However, when combined with simulated annealing and tabu search, the LP-based heuristic's performance can be improved.

A variant of MLDLSP, called the Multi-Level Proportional Lot-Sizing and Scheduling Problem (MLPLSP), in which at most two products (instead of one in the MLDLSP case) can be produced in each period, is studied by Kimms [51] and Kimms [52]. In Kimms [51], a single-machine system is examined. Two heuristic approaches are proposed: *radomized regrets*-based and tabu search approaches. The basic idea of the randomize regrets is to generate a large number of different production plans. The feasible solution with lowest cost is chosen. Each production plan is generated without using any information obtained from previous plans. According to the computational study, both heuristics perform equally well in terms of run-time and solution quality. A multi-level lot-sizing for multi-machine systems is studied by Kimms [52]. This is a direct extension of Kimms [51]. With slight modifications of the MIP model presented in Kimms [51], an MIP model for the extended problem is presented. A genetic algorithm is developed to solve the problem. The author reports that the proposed algorithm dominates the tabu search procedure, which is a straightforward extension of the one described in [51], both in terms of run-time performance and the ability to find feasible solutions.

5.6 Forecast and Planning Horizons

A *rolling schedule* is a cyclical procedure that starts with solving the model and implementing only the first period's decisions, then, based on the currently updated information, resolves the model. Baker [7] studies the effectiveness of the rolling schedules in production planning. The results from experiments suggest that rolling schedules are cost effective. Rolling schedule practice is widely accepted among decision makers because they are fully aware that the information (forecast) of the future tends to be inaccurate. Two relevant research questions that distinguish this line of research from others are:

- *Do the distant forecasts have an impact on the initial decisions?*
- *If not, how far off in the future of the forecast that has negligible impact on the initial decisions?*

There are some important terms that need to be introduced. First, a *Forecast horizon* is a finite horizon that is far enough that the information beyond it has no effect on the optimal decisions in the initial l periods. Second, a *Planning horizon* is the initial l periods. The planning horizon is sometimes called *decision horizon* (Bes and Sethi [14]). It should be noted that, in order to prove the existence of a planning horizon, one only needs to show that there is a valid forecast horizon for the case of $l = 1$.

Wagner and Whitin [97] introduced the notion of planning horizon in their *Planning Horizon Theorem*. It basically states, in part, that if periods 1 through t' are considered by themselves and it is optimal to incur a set up cost in period t' , then the optimal program for periods 1 through $t' - 1$ is also optimal for the T period model without foregoing optimality. Under this circumstance, the horizon starting from period 1 through $t' - 1$ is said to be a planning horizon, since any information from the periods beyond period $t' - 1$ has no effect on the optimal decisions of the first initial $t' - 1$ periods. The *Wagner-Whitin* model is also studied by Chand and Morton [23]. They develop procedures to find planning horizons of any length $l \leq T$, by using minimal forecast horizons. A variant of the *Wagner-Whitin* model in which demand and cost parameters are constant for an initial few periods is studied by Chand et al. [24]. Upper bounds of the forecast horizons are presented. They show that forecast horizons exist due to the presence of the setup cost even when there is no discounting. Blackburn and Kunreuther [20] extend the *Wagner-Whitin* model such that backlogging is allowed. They identify a sufficient (but not necessary) conditions for the existence of a forecast horizon. Federgruen and Tzur [32] also consider a dynamic lot-sizing model with backlogging, in which they provide necessary and sufficient condition for the existence of the forecast horizon. A review of forecast horizon results can be found in Federgruen and Tzur [33].

Bean and Smith [9] consider the general class of *discounted* problems involving sequential decision making over an infinite horizon. They address the existence issue of the planning horizon by providing necessary conditions under which the planning horizon is guaranteed to exist. The framework for the concepts of forecast and planning horizons is developed by Bes and Sethi [14] for the discounted dynamic stochastic optimization problems. Sufficient conditions for the existence of forecast horizons are also provided. They also develop stopping rules and computational procedures to obtain the horizons. With the assumption of constant parameters and demand in the initial few periods, Chand et al [25] extend Chand et al. [24] to the discounted case. They prove that a bound on the forecast horizon derived by [24] for the undiscounted problem is also valid for the model with a discount factor strictly less than one. Also, it is shown that discounting has influence on the length of the minimal forecast horizon.

5.7 Summary

In this chapter, we reviewed a wide range of the Dynamic Lot-Sizing Models (DLSM). For the Single-Level Single-Item Models, many efficient algorithms, such as Wagner and Whitin [97], Federgruen and Tzur [31], Wagelmans et al. [96], and Aggarwal and Park [3], have been developed. These algorithms can solve the problem in polynomial time.

Single-Level Multi-Item Models and Multi-Level Models, however, pose more computational challenges. Most of these problems are \mathcal{NP} -hard. Bitran and Yannasse [19] show that a two-item model could prove to be \mathcal{NP} -hard, even though its single-item version can be solved in polynomial time. A variety of approaches, such as Relaxation-Based Approaches, Variable Redefinition Approaches, and Cutting Plane Approaches, have been used to attack these difficult problems. None of these approaches prove to be efficient for every problem.

In this research, we consider the Capacitated Multi-Item Lot-Sizing Problem with Backorder and Integral Production Levels (CMBI). Having the integrality constraints makes the problem even harder. Even though many production systems operate under this restriction, to the best of our knowledge, there has been no attempt to seriously attack this problem.

We propose an MIP-Based solution method that efficiently solves CMBI. The solution method first solves the relaxed problem, called CMB, and uses the CMB's solution as an input to come up with the CMBI's solution. Chapter 6 describes how CMB can be solved efficiently using valid inequality selection procedures. Chapter 7 provides a detailed description of the MIP-Based solution method used to solve the CMBI.

Chapter 6

Solving the Capacitated Multi-Item Lot-Sizing Problem with Backorder (CMB) Using Valid Inequalities

6.1 Chapter Overview

This chapter introduces a supplier's cost determination model represented by the Capacitated Multi-Item Lot-Sizing Problem with Backorder (CMB). The model is used to determine the cost of supplying a *tuple*. It is useful to present the definition of a *tuple* again here. It was first given in Definition 4.1 on page 20.

Definition 6.1 (Tuple) Let $\mathcal{I} = \{1, \dots, N\}$ be a set of items the OEM demands from the suppliers and $\mathcal{D} = \{1, \dots, D\}$ be a set of the items' allowable delivery dates. A *tuple*, τ , is defined by:

$$\tau = (\mathcal{S}, (i, k_i) : \forall i \in \mathcal{S})$$

where $\mathcal{S} \subseteq \mathcal{I}$ and $k_i \in \mathcal{D}$ is the delivery date of item i .

Definition 6.2 (Product Type) Let \mathcal{S} be any subset of \mathcal{I} . Items in \mathcal{S} are said to be of the same *product type* if they require the same manufacturing process. We define $\mathcal{G} = \{1, \dots, G\}$ to be a set of product types of the items in \mathcal{I} .

This chapter is organized as follows. Section 6.2 describes CMB and its MIP formulation. The solving method that we will use is to add appropriate valid inequalities to the root node problem of the branch-and-bound tree. A family of valid inequalities, called the Lot-Sizing with Backorder (LSB) inequalities, is introduced in Section 6.3. In Section 6.4, we

then propose three strategies for selecting the LSB inequalities to be added to the existing constraints of the root node's problem. Combining our LSB inequalities selection strategies with the solving strategies provided by the CPLEX solver, we come up with twelve solving strategies. The performance of these twelve strategies is compared and the results are reported in Section 6.5.

6.2 The Model

We consider a situation where the supplier has a single production facility, which could be thought of as a single machine, a cell of machines, or even a whole production plant. The production facility operates in periods. A set of product types $\mathcal{G} = \{1, \dots, G\}$ can be produced by this facility. We assume that the OEM does not accept early delivery. Hence, early completed products have to be stored, and because of that, the inventory cost is incurred. Further, we assume that the late delivery is acceptable to the customers, but with a penalty charged to the supplier. The inventory cost and the late penalty per unit of product type g per period are constant and denoted by h_g and b_g respectively. Each product g requires a constant production time of p_g . The total production time of the facility in each period is limited by the capacity limit C_t . A fixed setup cost f_g is incurred in a period if in that period product g is produced. Setup time is assumed negligible.

A set of production periods is denoted by, $\Omega = \{1, \dots, T\}$, where T is the planning horizon and assumed very large. When the OEM initiates the auction, the supplier has already had some *committed demand*, $\underline{\Delta}$, which is a $G \times T$ matrix. Each element in $\underline{\Delta}$, denoted by \underline{d}_{gt} , represents the demand quantity for product type g on delivery date t . Each tuple can be considered as *tentative demand*. We can easily transform tuple τ to the *tentative demand matrix*, $\widehat{\Delta}$, in which each element, \widehat{d}_{gt} , represents tentative demand quantity for product g on delivery date t .

The *total demand matrix*, Δ , therefore can be defined by:

$$\Delta = \underline{\Delta} + \widehat{\Delta}$$

and obviously, each element in Δ , which is denoted by d_{gt} , is defined by:

$$d_{gt} = \underline{d}_{gt} + \widehat{d}_{gt}$$

We now define the decision variables of CMB. The production level of product g in period t is denoted by x_{gt} . A binary variable, y_{gt} indicates whether in period t product g is produced.

$$y_{gt} = \begin{cases} 1, & \text{if } x_{gt} > 0 \\ 0, & \text{otherwise} \end{cases}$$

The quantity of product g in period t that is carried over from the previous period is denoted by s_{gt} . The quantity of product g that is backordered in period t is denoted by u_{gt} .

The following MIP formulation is the CMB model used to determine the total cost of the total demand, $Z(\Delta)$ (Note that the total demand, Δ includes the tentative demand, $\widehat{\Delta}$, which is transformed from the tuple).

$$\text{(CMB)} \quad Z(\Delta) = \min \sum_{g \in \mathcal{G}} \sum_{t \in \Omega} (f_j y_{gt} + h_j s_{gt} + b_j u_{gt}) \quad (6.1)$$

subject to

$$x_{gt} + s_{g(t-1)} + u_{gt} = d_{gt} + s_{gt} + u_{g(t-1)}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (6.2)$$

$$x_{gt} \leq \left(\frac{C_t}{p_j}\right) y_{gt}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (6.3)$$

$$\sum_{g \in \mathcal{G}} x_{gt} p_j \leq C_t \quad \forall t \in \Omega \quad (6.4)$$

$$\sum_{t \in \Omega} x_{gt} = \sum_{t \in \Omega} d_{gt}, \quad \forall g \in \mathcal{G} \quad (6.5)$$

$$y_{gt} \in \{0, 1\}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (6.6)$$

$$x_{gt}, s_{gt}, u_{gt} \geq 0, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (6.7)$$

Obviously, if we replace d_{gt} with \underline{d}_{gt} , the above formulation determines the cost for the committed demand, $Z(\underline{\Delta})$. Therefore, the cost of a tuple τ on supplier j , denoted by $c_j(\tau)$, can be defined by:

$$c_j(\tau) = Z(\Delta) - Z(\underline{\Delta}) \quad (6.8)$$

Recall from Part I that we assumed that each supplier j in \mathcal{N} has private knowledge about his costs on all possible tuples. Given that the supplier's cost determination model can be represented by CMB, his cost on a tuple, $c_j(\tau)$, is calculated by equation (6.8).

Let us now consider the CMB formulation in detail. The objective function (6.1) is to minimize the total setup cost, inventory holding cost, and late penalty cost. Equation (6.2) represents inventory balance constraints. The setup state variable y_{gt} is controlled by the logical constraints (6.3). Inequality (6.4) limits the maximum total production of a period. Constraints (6.5) ensures that, eventually, all demands will be fulfilled. Constraints (6.6) restrict the setup state variables y_{gt} to be binary number, while constraints (6.7) enforce production, inventory and backlogging levels to be nonnegative. The solution set satisfying (6.2) – (6.7) is called \mathcal{X}^{CMB} .

If $C_t = +\infty$ for all t (uncapacitated case), then there exist several solution procedures that are able solve the problem in polynomial time. For the single-item constant capacity case ($G = 1$ and $C_t = C$ for all t), there exists a polynomial time dynamic programming algorithm that solves the problem. However, Wolsey [100] showed that a lot-sizing problem becomes \mathcal{NP} -hard even for the single-item case if capacities are non-constant.

6.3 Valid Inequalities

We first begin this section by introducing the definition of a valid inequality. Informally, an inequality is called *valid inequality* for a solution set \mathcal{X} if it does not exclude any possible solution in \mathcal{X} . The formal definition is given below.

Definition 6.3 Let π and π_0 be row vector data and \mathbf{x} be a column vector variable. An inequality $\pi\mathbf{x} \leq \pi_0$ is *valid* for a solution set \mathcal{X} , if $\pi\mathbf{x} \leq \pi_0$ for all $\mathbf{x} \in \mathcal{X}$.

The term valid inequality is used here to refer to a constraint based on problem specific structure and derived only from the formulation. Cutting planes, or cuts, on the other hand, usually refer to constraints generated by a generic algorithm after solving an LP relaxation (See Magee and Glover [66]). However, both valid inequalities and cuts share the same

purpose of cutting off portions of the feasible region after the LP relaxation is solved. The distinction in the two terms will not be taken as an important issue here. More theoretical discussions about valid inequality can be found in Nemhauser and Wolsey [78].

Usually, significant amount of computing effort is required in solving an MIP problem. But if the convex hull of the feasible set \mathcal{X} of the MIP problem, denoted by $\text{conv}(\mathcal{X})$, can be defined, we can solve the LP relaxation described by $\text{conv}(\mathcal{X})$ and obtain the MIP optimal solution right away. However, in most practical problems, it is not always possible to compute $\text{conv}(\mathcal{X})$.

Given that the single-item non-constant capacity lot-sizing problem is \mathcal{NP} -hard (See Wolsey [100]), it can be inferred that our problem, with multiple products, is also \mathcal{NP} -hard. Therefore, unless $\mathcal{P} = \mathcal{NP}$, it is difficult to fully determine $\text{conv}(\mathcal{X}^{CMB})$.

Barany et al. [8] reformulated the multi-item capacitated lot-sizing problems using a class of valid inequalities, called the (l, S) inequalities, which are facets for the single-item uncapacitated problem. The objective of their approach is to obtain a good approximation of the convex hull of the solution set. Their lot-sizing model, however, does not consider backlogging.

Agra and Constantino [4] investigated the optimal properties of the solution to the uncapacitated lot-sizing problems with backlogging. They were able to prove that in the optimal solution, a demand will be exclusively satisfied either by stock, by production in that period, or by backlog. Also, they found that if the machine is setup in a given period, the demand of that period is satisfied by production in that period. With these properties, they developed valid inequalities for uncapacitated lot-sizing problems.

Following, we present a family of valid inequalities for CMB. In this research, the plan is to add a subset of these valid inequalities at the root node of the branch-and-bound tree.

Proposition 6.1 The inequality

$$x_{gt} \leq d_{gt}y_{gt} + s_{gt} + u_{g(t-1)}, \quad g \in \mathcal{G}, t \in \Omega \quad (6.9)$$

is valid inequality for \mathcal{X}^{CMB} .

Proof We prove the proposition by considering two possible cases.

Case 1 ($y_{gt} = 0$) If $y_{gt} = 0$ we can conclude that $x_{gt} = 0$ also. The inequalities can now be rewritten for this situation as $0 \leq 0 + s_{gt} + u_{g(t-1)}$. Obviously, the inequalities are valid because $s_{gt} \geq 0$ and $u_{gt} \geq 0$ for all $g \in \mathcal{G}$ and for all $t \in \Omega$.

Case 2 ($y_{gt} = 1$) In this situation, the inequalities can be rewritten as $x_{gt} \leq d_{gt} + s_{gt} + u_{g(t-1)}$. Recall that we have the inventory balance constraints $x_{gt} + s_{g(t-1)} + u_{gt} = d_{gt} + s_{gt} + u_{g(t-1)}$. It is clear that $x_{gt} \leq d_{gt} + s_{gt} + u_{g(t-1)}$ must hold, hence the inequalities are again valid. \square

It is possible to define a larger set of valid inequalities for CMB.

Proposition 6.2 For any $S \subset \{1, \dots, t\}$, the inequality

$$\sum_{t \in S} x_{gt} \leq \sum_{t \in S} d_{gt} + s_{gt} + \sum_{t \in S} u_{gt}, \quad g \in \mathcal{G}, t \in \Omega \quad (6.10)$$

is valid inequality for \mathcal{X}^{CMB} .

These valid inequalities are a generalization of the (l, S) inequalities, and the proof of validity is similar. One code that uses these inequalities to solve lot-sizing problems is *bc-prod* (Belvaux and Wolsey [10]).

Taking implementation requirements into consideration, we focus on valid inequalities (6.9). Our results in Section 6.5 show that the inequalities are effective in solving CMB. Because this family of valid inequalities is derived from the lot-sizing problem with backorders, we will call the valid inequalities shown in inequality (6.9) as *LSB inequalities*. Furthermore, since a valid inequality can be identified by two indices g and t , we will denote by $\text{LSB}(\tilde{g}, \tilde{t})$ an LSB inequality with $g = \tilde{g}$ and $t = \tilde{t}$.

6.4 Proposed Valid Inequality Selection Strategies

The Branch-and-bound algorithm is one of the most prominent MIP solution algorithms. When used, it guarantees an optimal solution if the problem is solvable and sufficient computing time is allowed. During the solving process, if an integer solution is found, the

Branch-and-Bound algorithm provides optimality gap information, indicating the percentage difference between the current integer solution and the best possible solution. The strategy of branch-and-bound is to search through a tree consisting of nodes, each of which represents an LP relaxation of the original MIP. If the solution found at a node contains fractional variables, one fractional variable is selected and two child nodes are generated. The two children essentially represent the augmented problems of the parent node. The first child node is the problem which is augmented by a constraint restricting the upper bound of the selected variable to be at the floor value of the current solution value. The second child node is the problem augmented by a constraint restricting the lower bound of the selected variable to be at the ceiling value of the current solution value. This process is referred to as the *branch* step. A node is closed if it yields an integer solution. Considering a minimization MIP problem, a node is also closed if its objective value is greater than the best known integer solution's objective value. The algorithm stops when there is no open node left in the branch-and-bound tree.

Another solution method that some researchers have been using successfully is the *Branch-and-Cut* algorithm. In fact, branch-and-cut is adapted from the branch-and-bound algorithm by adding valid inequalities (also called *cuts* or *cutting planes*) at some or all of the nodes in the tree. If cuts are added only at the root node, the algorithm is then called *Cut-and-Branch*. The philosophy of branch-and-cut is to spend more time at each node to tighten the LP relaxation formulation so that the lower bounds in the tree are increased. Once the lower bounds are increased, the gap between the lowest lower bound and the smallest integer solution value, in general, will be decreased. This will, in effect, reduce the tree size that the algorithm has to search through.

Branch-and-cut, nevertheless, has some potential drawbacks. Branch-and-cut can be an inefficient solution method if the cuts added are not too strong, in the sense that they do not eliminate a lot of fractional points. In that case, the size of the tree will not be much reduced. Additionally, branch-and-cut could be very slow if the computing time required at each node for the addition of cuts is too high. In our experience, adding cuts to all the

nodes in the tree can occasionally result in a dramatic increase in the computing time. As a consequence, it will be seen later that, in our implementation, we choose to add the LSB inequalities at the root node only.

By adding valid inequalities to the model, we increase the problem size. Hence, the computing time required to solve the problem would generally increase as a result. Careful consideration as to which valid inequalities to add to the model needs to be taken. Three valid inequality selection strategies are developed and described in detail in Section 6.4.1–6.4.3. These selection strategies can be applied to any family of valid inequalities. In this research, however, we apply them only to the LSB inequalities.

6.4.1 All Optimal Point Violated Cuts Strategy (All-V)

Let RCMB and $(\bar{x}^*, \bar{y}^*, \bar{s}^*, \bar{u}^*) = (\bar{x}_{11}^*, \dots, \bar{x}_{gt}^*, \bar{y}_{11}^*, \dots, \bar{y}_{gt}^*, \bar{s}_{11}^*, \dots, \bar{s}_{gt}^*, \bar{u}_{11}^*, \dots, \bar{u}_{gt}^*)$ be the LP relaxation of CMB and its optimal solution, respectively. This valid inequality selection strategy follows two steps described below.

All-V:

Step 1 Solve RCMB and obtain $(\bar{x}^*, \bar{y}^*, \bar{s}^*, \bar{u}^*)$

Step 2 Add to CMB all the LSB inequalities

$$x_{gt} \leq d_{gt}y_{gt} + s_{gt} + u_{g(t-1)}, \quad g \in \mathcal{G}, t \in \Omega$$

such that

$$\bar{x}_{gt}^* > d_{gt}\bar{y}_{gt}^* + \bar{s}_{gt}^* + \bar{u}_{g(t-1)}^*$$

Let $v(j, t) = \bar{x}_{gt}^* - (d_{gt}\bar{y}_{gt}^* + \bar{s}_{gt}^* + \bar{u}_{g(t-1)}^*)$ be the violation of the LP optimal point $(\bar{x}^*, \bar{y}^*, \bar{s}^*, \bar{u}^*)$ on the LSB(j, t). This selection strategy adds *all* the LSB(j, t)'s that have $v(j, t) > 0$ to CMB.

Figure 6.1 illustrates a simple example of a node's LP feasible region, shown as the shaded area, and three LSB inequalities. If F is the LP optimal solution found in Step 1, the All-V

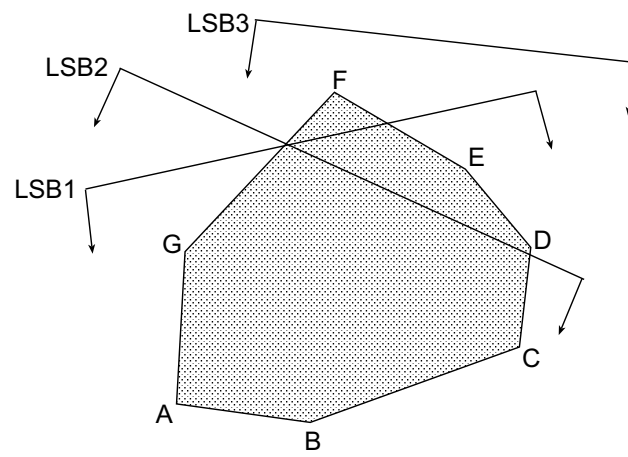


Figure 6.1 The LSB inequality selection in All-V, Most-V, and Max-XP

strategy will add both of the LSB1 and LSB2 inequalities to the model because they are violated by F. The LSB3 inequality, on the other hand, will not be added since it is not violated.

One major benefit of using All-V is that it requires a relatively modest amount of computing time. This is because there is only one LP problem (i.e., the RCMB) solved (step 1). Furthermore, in Step 2, we have to determine if an inequality is violated by the LP solution. So, for LSB inequalities, only $G \times T$ calculations are required for this step. However, if we apply this selection strategy to other cut families in which the number of cuts grows exponentially with the problem size, this step could prove to be impractical.

In the process of adding valid inequalities, it is hoped that the added inequalities will cut off a large number of vertices so that the lower bound would be significantly improved. But since All-V selects all the $\text{LSB}(j, t)$'s with $v(j, t) > 0$, some of the $v(j, t)$'s could be very small. These $\text{LSB}(j, t)$'s with very small $v(j, t)$'s are unlikely to be able to cut off a large number of fractional points. Adding these small violation inequalities, therefore, could result in the increase in problem size without really improving the lower bound. Realizing this possible shortcoming, we develop the following valid inequality selection strategy.

6.4.2 Most Violated Strategy (Most-V)

After all the violated inequalities are added to the model, the LP relaxation of the augmented CMB is solved for the LP optimal solution $(\bar{x}^*, \bar{y}^*, \bar{s}^*, \bar{u}^*)$. It is not guaranteed, however, that $(\bar{x}^*, \bar{y}^*, \bar{s}^*, \bar{u}^*)$ will contain a large number of integer values. In this section, we present a valid inequality selection strategy that tries to find as many integer values in the root node's optimal solution as possible. We hope that the resulting \mathcal{X}^{CMB} description will represent a good approximation of $\text{conv}(\mathcal{X}^{CMB})$.

Most-V follows an iterative procedure. Let RCMB be the CMB relaxation to which valid inequalities will be added. Also, let $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$ and v^k indicate the LP optimal solution and the maximum violation found in the k -th iteration of the loop, respectively. Most-V can be described as follows.

Most-V:

Step 0 Let $k = 0$.

Step 1 Increment k , solve RCMB, and obtain $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$

Step 2 For all $g \in \mathcal{G}$ and $t \in \Omega$, find

$$v^k = \max_{g \in \mathcal{G}, t \in \Omega} \{\bar{x}_{gt}^{k*} - (d_{gt}\bar{y}_{gt}^{k*} + \bar{s}_{gt}^{k*} + \bar{u}_{g(t-1)}^{k*})\}$$

and the corresponding

$$(g', t') = \arg \max_{g \in \mathcal{G}, t \in \Omega} \{\bar{x}_{gt}^{k*} - (d_{gt}\bar{y}_{gt}^{k*} + \bar{s}_{gt}^{k*} + \bar{u}_{g(t-1)}^{k*})\}$$

Step 3 If $v^k \leq 0$, STOP

Step 4 Else, add $\text{LSB}(g', t')$ to RCMB

Step 5 Go back to Step 1

Basically, Most-V adds the single most violated LSB inequality in each loop iteration (step 1 to step 5). From Figure 6.1, suppose F is the LP optimal solution obtained in Step 1, Most-V will then consider the LSB1 and LSB2 inequalities. Most-V also considers LSB3, but since it is not violated, we will not discuss it here. Whichever inequality is more violated will be added. The looping terminates at Step 3 of loop iteration k , if there is no LSB inequality that is violated by the most recent optimal solution ($v^k \leq 0$).

From our experiments, it has been shown that while most of the variables in the first few loops take fractional values, only few of them remain fractional when Most-V terminates. This would certainly substantiate our conjecture that Most-V could be used to find a good approximation of $\text{conv}(\mathcal{X}^{CMB})$.

Some drawbacks of Most-V do exist, however. By using Most-V, at any loop k , the corresponding RCMB has to be solved and the LSB that is most violated by $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$ is selected. If there are a large number of LSB inequalities to consider and most of them

can be added according to the Most-V method, the computing time required to run Most-V could be significant. Another possible drawback of Most-V, and as well as All-V for that matter, is that by cutting off an LP optimal point with a valid inequality, even more extreme points can potentially be created. Therefore, if the LP is solved using the Primal Simplex Method, which has to move from one extreme points to another until optimal point is found, the computing time to solve the LP in the children nodes would also increase as a result. Another reason for increased time would be increased problem size due to more constraints and variables.

6.4.3 Maximum Number of Extreme Points Cut Off Strategy (Max-XP)

We present a valid inequality selection strategy called Maximum Number of Extreme Points Cut Off (Max-XP), by which the LSB inequalities are selected based on the number of extreme points they can cut off.

The number of the extreme points cut off is counted backward starting from the LP optimal point. This is done with two purposes. Given the fact that the Simplex algorithm moves from one extreme point to another with improving solution quality, the points close to the optimal solution are likely to be of high quality. Recall that a high quality solution for a minimization problem means a low objective value. The first purpose of the Max-XP method, therefore, is to exclude high quality non-integer solutions from the feasible region. If only low quality solutions are left in the feasible region, we can achieve higher lower bound, which in turn will reduce the number of nodes need to be processed.

The second purpose is that, with a valid inequality that could cut off a lot of extreme points, it is more likely that the effect of a new valid inequality creating new fractional points will be alleviated. Less iterations will be required to solve the LP in the child nodes.

We summarize the notation used in Max-XP in Table 6.1 on page 82. Now, we present the Max-XP valid inequality selection strategy.

Max-XP:

Notation	Description
i	number of extreme points counted
k	number of valid inequalities being added
RCMB	CMB relaxation to which valid inequalities are added
I^k	number of iterations needed to solve $\bar{\mathcal{P}}^k$
$\text{LSB}(j, t)$	a valid inequality defined by (j, t)
$(\bar{x}^{ki}, \bar{y}^{ki}, \bar{s}^{ki}, \bar{u}^{ki})$	an i -th extreme point in k -th loop
P	number of extreme points to be kept
σ^k	a stack keeping extreme points information object in k -th loop
v_{gt}^{ki}	violation of the $\text{LSB}(j, t)$ by $(\bar{x}^{ki}, \bar{y}^{ki}, \bar{s}^{ki}, \bar{u}^{ki})$
v_{gt}^k	total sum of v_{gt}^{ki} 's
γ_{gt}^k	number of extreme points $\text{LSB}(j, t)$ can cut off in the k -th loop
γ_{max}^k	maximum number of extreme points a valid inequality can cut off in the k -th loop
α^k	an array of information about the valid inequalities that can cut off maximum number of extreme points in k -th loop

Table 6.1 Notation necessary to describe Max-XP solution method

Step 0 Let $k = 0$ and $i = 0$.

Step 1 Set $\gamma_{max}^k = 0$, solve RCMB, obtain $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$ and I^k .

Step 2 Resolve RCMB, but with iteration limit of $I^k - P$.

Step 3 For each $i = 0$ to $i = P - 1$, increment i , continue solving RCMB with iteration limit of one, and *push* the extreme point information $(\bar{x}^{ki}, \bar{y}^{ki}, \bar{s}^{ki}, \bar{u}^{ki})$ in a stack σ^k .

Step 4 Reset $i = 0$ for the next loop.

Step 5 For each $g \in \mathcal{G}$ and $t \in \Omega$, *pop* extreme point information object from σ^k one by one, and for each i extreme point information object, calculate

$$v_{gt}^{ki} = \bar{x}_{gt}^{ki} - (d_{gt}\bar{y}_{gt}^{ki} + \bar{s}_{gt}^{ki} + \bar{u}_{g(t-1)}^{ki})$$

then add v_{gt}^{ki} to v_{gt}^k by,

$$v' = v_{gt}^k$$

$$v_{gt}^k = v' + v_{gt}^{ki}$$

and if $v_{gt}^{ki} > 0$, increment γ_{gt}^k . Also, if $\gamma_{gt}^k > \gamma_{max}^k$, set $\gamma_{max}^k = \gamma_{gt}^k$ and keep (j, t) as an object in array α^k .

Step 6 If $|\alpha^k| = 0$, STOP. Else if $|\alpha^k| = 1$ where (g', t') is the only element in α^k , add $\text{LSB}(g', t')$ to CMB and increment k . But else, if $|\alpha^k| > 1$, find $(\hat{j}, \hat{t}) = \arg \max\{v_{gt}^k\}$ then add $\text{LSB}(\hat{j}, \hat{t})$ to RCMB and increment k .

Step 7 Go back to Step 1.

At Step 1, the (augmented) CMB's LP relaxation is solved in order to determine the number of iterations used. Then, at Step 2, the problem is resolved but this time it stops at P iterations before reaching RCMB optimal solution. Step 3 continues solving the problem one iteration at a time until optimality is attained. During this step, the extreme points information $(\bar{x}^{ki}, \bar{y}^{ki}, \bar{s}^{ki}, \bar{u}^{ki})$ is collected. At Step 5, Max-XP goes through all the $\text{LSB}(j, t)$'s

to determine γ_{max}^k and v_{gt}^k . The selection and addition of the valid inequality is accomplished at step 6. This process will go on until, in the k -th loop, there is no valid inequality that can cut off the optimal point $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$. The result of running Max-XP is the augmented RCMB in which a subset of LSB inequalities are added.

Referring to Figure 6.1, suppose that, in the k -th loop, the Simplex algorithm finds the first feasible solution at point A and follows A-B-C-D-E-F path, where F is the LP optimal found in Step 1. Two inequalities that are violated by F are LSB1 and LSB2. Based on the LSB inequalities selection criteria used by Max-XP, LSB2 will be chosen and added to the model because it can cut off three extreme points, namely points F, E, and D. The LSB1 inequality is not chosen because it can cut off only the LP optimal point F. As can be seen in Figure 6.1, once LSB2 is added, it reshapes the LP feasible region in such a way that the number of extreme points is reduced. When the LP is run again at the child nodes, therefore, it is likely that the number of iterations needed to solve the LP's will decrease. Moreover, with the exclusion of the extreme points D, E, and F from the feasible region, the Simplex algorithm has to deviate from the previous path. If those D, E, and F points represent the best, the second best, and the third best solution, respectively, it means that the next time RCMB is solved, the solution will not be better than the third best one. In other words, using Max-XP could potentially lead to a much higher lower bound (for the minimization problem) and hence a much smaller optimality gap.

Max-XP strategy is designed to serve two purposes, as mentioned earlier, in a very straightforward manner. However, it usually takes much more time to select a valid inequality to be added to the model than other strategies. This could prove to be inefficient especially in solving small problems where such overhead is deemed too high.

The reasons that Max-XP takes more time than other strategies are described as follows. Obviously, Max-XP iteratively solves RCMB, as Most-V does. But unlike Most-V, Max-XP needs to solve RCMB twice for each addition of a valid inequality, the first time to determine the total number of iterations needed, and the second time for the last P extreme points. Moreover, by using Max-XP, since the information regarding the last P extreme

points has to be kept, a tremendous proportion of the time will be used in the input/output process. Lastly, in order to find the total sum of violations in any loop, Max-XP has to do more calculations than All-V and Most-V do. While Most-V does a single v_{gt}^k calculation for each $\text{LSB}(j, t)$, Max-XP has to do P v_{gt}^{ki} calculations before being able to determine the total sum of violations v_{gt}^k . In our experience with Max-XP, it is observed that most of the time is spent on the input/output process at Step 3. Improvement in the algorithm at the input/output step and/or the improvement of the input/output technology would lead to a better performance of Max-XP solution method.

6.5 Implementations and Computational Results

Branch-and-Bound is the most widely used algorithm for solving MIP problems. It is natural for us to solve CMB using the Branch-and-Bound algorithm. Many commercial MIP solvers are available. Most of them provide users with a number of options. For MIP, users can specify the types of generic cuts and the nodes in the branch-and-bound tree at which cuts will be added.

6.5.1 Solving Strategies Considered

In this research, we formulate all the problems in GAMS modeling language and solve them by calling the CPLEX solver. The General Algebraic Modeling System (GAMS) is a high-level modeling language for mathematical programming problems. Once a problem is formulated, the user has to identify the solver that will be called to actually solve the problem.

In solving an IP or MIP problem, CPLEX, by default, will automatically determine which types of generic cuts are to be added and the nodes in the branch-and-bound tree at which the cuts will be added. While the default settings work well in general cases, users are provided with solving options in the case where some adjustments are needed. With regard to the cuts generated by CPLEX, we consider two mixed 0-1 cut families, namely *cover cuts*

and *flow cover cuts*. Descriptions of these two cuts can be found, for example, in Avriel and Golany [66] and Wolsey [100]. These cuts will be simply called *CPLEX cuts* from now on.

In order to let users control the addition of the CPLEX cuts, CPLEX provides users with four options: not adding any of the cuts at all, adding the cuts at the root node only, adding the cuts at every node throughout the tree, and lastly, adding the cuts at any nodes throughout the tree as CPLEX sees beneficial. Since adding the cuts throughout the tree, especially for the large-sized problems, requires tremendous computational effort, in our experiment, we consider only the following three GAMS/CPLEX settings:

- None of the CPLEX cuts are added at all (No-C)
- CPLEX cuts can be added at the root node only (C-Root)
- CPLEX cuts can be added at any nodes throughout the branch-and-bound tree as and when CPLEX finds it useful (C-Def)

The last setting is in fact the default setting by CPLEX.

With regard to the LSB inequalities, we have four implementation options. Three of them are the All-V, Most-V, and Max-XP strategies presented in Section 6.4.1– 6.4.3. They add LSB inequalities only at the root node. The last option is not to add any of the LSB inequalities at all (No-LSB). Since there are three options in the addition of the CPLEX cuts and four options in the addition of the LSB inequalities, we come up with the total of twelve solution strategies in solving CMB.

We denote α/β the solution strategy that employs α CPLEX cut option and β LSB inequalities selection scheme. For example, C-Root/Most-V represents the solution strategy that adds CPLEX cuts at the root node and adds the LSB inequalities through Most-V strategy. It should be noted here also that, in the experiment, for the Max-XP method, we keep ten extreme points ($P = 10$).

A main program, which is coded in Perl scripting language, is created to manage all the implementations and experiments. One of its responsibilities is to implement All-V, Most-V,

and Max-XP. The selected LSB inequalities are added to the GAMS model on-the-fly. The program will configure the CPLEX cut option according the solving strategy being used and call GAMS/CPLEX to solve the problem. All the implementations are done on a Pentium III 800 MHz PC machine with Linux operating system.

6.5.2 Test Problems

We consider a manufacturing system being able to produce five product types ($|\mathcal{G}| = 5$). Each unit of a product type g requires a constant processing time p_j . In the experiment, we pick p_j from a uniform distribution of the range $[\bar{p}_l, \bar{p}_h]$, where \bar{p}_l and \bar{p}_h are constant numbers and $\bar{p}_l < \bar{p}_h$. It is assumed that the system has already committed to some of demand in the past. The committed demand in a period t is non-negative, and it can be written as a linearly decreasing function over time

$$D(t) = \min\{0, D_1 - \delta t\} \quad (6.11)$$

where D_1 and $D(t)$ are the total demands (in integral units) of all the five products in period 1 and period $t > 1$, respectively. Please note that $D(t)$ can also be expressed as $D(t) = \sum_{j=1}^5 d_{j,t}$. To assign a value to a $d_{j,t}$, we pick a random number r_j in $(0, 1)$ for each product type g , then assign

$$d_{j,t} = \text{round}\left(\frac{r_j}{\sum_{j=1}^5 r_j} D(t)\right) \quad (6.12)$$

where the $\text{round}(\bullet)$ function returns the closest integer to the argument. We restrict all of the demands to be integer so that problems generated here can be used in the next chapter, where production levels have to be integral values. Capacity limits (in units of time) of the system are selected from a uniform distribution of the range $[\bar{C}_l, \bar{C}_h]$, where $\bar{C}_l < \bar{C}_h$ and they are constants. For any product type g , the late penalty b_j , the holding cost h_j , and a setup cost f_j are picked from uniformly distributed ranges $[\bar{b}_l, \bar{b}_h]$, $[\bar{h}_l, \bar{h}_h]$, and $[\bar{f}_l, \bar{f}_h]$, respectively. In our experiment, we assign values to all of the above parameters and they are summarized in Table 6.2.

Parameter	Value	Parameter	Value	Parameter	Value
D_1	80	\bar{C}_l	120	\bar{h}_l	1
δ	4	\bar{C}_h	200	\bar{h}_h	5
\bar{p}_l	1	\bar{b}_l	10	\bar{f}_l	30
\bar{p}_h	5	\bar{b}_h	50	\bar{f}_h	150

Table 6.2 Parameter values used in the experiment

6.5.3 Computational Results

Ten test problems were generated. These problems are used to evaluate the twelve solving strategies described in Section 6.5.1. A solving process stops if one percent optimality gap is achieved, or if the maximum computing time limit (excluding the LSB inequalities selection time) of 1,800 seconds is reached. Performance criteria that we are interested in are *computing time*, *optimality gap achieved*, and *best objective found*.

6.5.3.1 Computing Time

We solve each of the ten test problems using the twelve solution strategies described in Section 6.5.1. The computing time used by each strategy is recorded. For a given problem, we compare the computing times used by the twelve strategies by normalizing all the computing times by the minimum one. Obviously, the fastest strategy for each problem will have the normalized computing time of one. Other slower strategies will take values greater than one.

Since the selection of the LSB inequalities could require significant computing time, especially for Most-V and Max-XP methods, we present the results in two separate sets of tables. Table 6.3 and Table 6.4 show the results when the LSB inequalities selection times used by All-V, Most-V, and Max-XP methods are not included in the total computing time. Table 6.5 and Table 6.6 show the results which include the LSB inequalities selection times used by All-V, Most-V, and Max-XP methods.

It can be observed from Table 6.3 and Table 6.5 that none of the ten test problems can be solved to one percent optimality gap within half an hour using No-C/No-LSB strategy. On the other hand, by using other solution strategies, the one percent optimality gap can be achieved in six of the ten problems, mostly within seconds or a few minutes. The asterisks (*) displayed in these two tables represent the case where a solution strategy cannot reach the one percent optimality gap within half an hour.

In Table 6.4 and Table 6.6 the symbol ‘n/a’ is used to denote the situation in which the problem cannot be solved in half an hour time using the strategy. Considering the normalized computing time in these tables, it is found that all of the fastest solution strategies add the LSB inequalities, either through All-V, Most-V, or Max-XP. Once the LSB inequalities selection time is taken into account, see Table 6.6, the fastest solution strategies are the ones that add LSB inequalities through All-V methods only. It can be inferred from the results that solving CMB by adding LSB inequalities through All-V is faster than solving the problem applying generic cuts alone.

If we compare the solution strategy that adds only CPLEX cuts at the root (C-Root/No-LSB) and the solution strategy that adds only LSB cuts at the root node through All-V strategy (No-C/All-V), it is clear that No-C/All-V solves the test problems faster than C-Root/No-LSB does. In fact, judging by the summation of the normalized computing times of the six test problems that can be solved within half an hour, No-C/All-V is the fastest solution strategy.

6.5.3.2 Optimality Gap Achieved

During the solving process, the optimizer keeps track of the best lower bound, denoted by LB , and the best integer objective, denoted by UB . Since LB represents the best possible objective that we can achieve, the smaller the difference between UB and LB , the better quality to solution is. This difference is called the *optimality gap*, γ , and it can be defined as

$$\gamma = \frac{UB - LB}{LB} \quad (6.13)$$

The optimality gaps achieved by the solution strategies are shown in Table 6.7. In the table, if an entry is greater than one percent, it means that the strategy cannot reach one percent optimality gap within half an hour. Without any cuts added (No-C/No-LSB), the optimality gap achieved is always higher than those achieved by other eleven strategies, in which some cuts are added. For problem 3-6, even though none of the strategies can reach one percent gap within half an hour, all of the other eleven strategies are able to achieve very small gaps, in comparison with those of No-C/No-LSB's.

6.5.3.3 Best Objective Found

Among the twelve solution strategies, it is also desirable to know if any one of the strategies generally gives better (lower) objective. With a lower objective, lower cost can be achieved, and the supplier has a better chance to win in the auction. The objective values achieved by the solution strategies are given in Table 6.8.

It shows in the table that the strategies that give the lowest objective in each problem add the LSB inequalities, either through All-V, Most-V, or Max-XP. If we compare objectives obtained from C-Root/No-LSB and No-C/All-V (the fastest strategy, see Section 6.5.3.1), in seven out of ten problems, No-C/All-V gives better objectives. Based on these ten problems, on average, strategy C-Def/Most-V provides the best (lowest) objective, while strategy No-C/No-LSB yields the worst (highest) objective.

Problem		1	2	3	4	5	6	7	8	9	10
No-C	No-LSB	1800*	1800*	1800*	1800*	1800*	1800*	1800*	1800*	1800*	1800*
	All-V	407	76	1800*	1800*	1800*	1800*	25	4	87	16
	Most-V	916	88	1800*	1800*	1800*	1800*	24	14	114	17
	Max-XP	693	41	1800*	1800*	1800*	1800*	24	2	113	18
C-Root	No-LSB	552	100	1800*	1800*	1800*	1800*	118	12	106	31
	All-V	175	88	1800*	1800*	1800*	1800*	72	15	153	14
	Most-V	875	85	1800*	1800*	1800*	1800*	72	15	117	14
	Max-XP	288	356	1800*	1800*	1800*	1800*	72	2	117	14
C-Def	No-LSB	548	75	1800*	1800*	1800*	1800*	119	18	97	29
	All-V	175	52	1800*	1800*	1800*	1800*	71	18	137	13
	Most-V	899	32	1800*	1800*	1800*	1800*	72	15	106	13
	Max-XP	288	120	1800*	1800*	1800*	1800*	72	2	107	13

Table 6.3 Computing time (seconds) excluding LSB valid inequalities selection time

Problem		1	2	3	4	5	6	7	8	9	10
No-C	No-LSB	10.27	56.96	n/a	n/a	n/a	n/a	74.84	923.08	20.73	136.78
	All-V	2.32	2.41	n/a	n/a	n/a	n/a	1.03	1.89	1.00	1.25
	Most-V	5.22	2.77	n/a	n/a	n/a	n/a	1.01	7.35	1.31	1.33
	Max-XP	3.95	1.31	n/a	n/a	n/a	n/a	1.00	1.09	1.31	1.35
C-Root	No-LSB	3.15	3.15	n/a	n/a	n/a	n/a	4.92	6.34	1.22	2.33
	All-V	1.00	2.79	n/a	n/a	n/a	n/a	3.00	7.48	1.76	1.07
	Most-V	4.99	2.70	n/a	n/a	n/a	n/a	2.99	7.50	1.35	1.07
	Max-XP	1.64	11.26	n/a	n/a	n/a	n/a	2.98	1.00	1.34	1.09
C-Def	No-LSB	3.13	2.36	n/a	n/a	n/a	n/a	4.97	9.38	1.11	2.23
	All-V	1.00	1.63	n/a	n/a	n/a	n/a	2.96	9.31	1.58	1.00
	Most-V	5.13	1.00	n/a	n/a	n/a	n/a	3.01	7.49	1.22	1.02
	Max-XP	1.64	3.79	n/a	n/a	n/a	n/a	3.00	1.00	1.23	1.01

Table 6.4 Normalized computing time excluding LSB inequalities selection time

Problem		1	2	3	4	5	6	7	8	9	10
No-C	No-LSB	1800*	1800*	1800*	1800*	1800*	1800*	1800*	1800*	1800*	1800*
	All-V	408	77	1801*	1801*	1801*	1801*	26	5	88	17
	Most-V	950	120	1832*	1842*	1830*	1835*	54	41	152	51
	Max-XP	885	233	1968*	2028*	1960*	1992*	180	150	317	198
C-Root	No-LSB	552	100	1800*	1800*	1800*	1800*	118	12	106	31
	All-V	176	89	1801*	1801*	1801*	1801*	73	16	154	15
	Most-V	910	17	1832*	1842*	1830*	1835*	101	42	155	48
	Max-XP	480	548	1968*	2028*	1960*	1992*	228	150	321	194
C-Def	No-LSB	548	75	1800*	1800*	1800*	1800*	119	18	97	29
	All-V	176	53	1801*	1801*	1801*	1801*	72	19	138	14
	Most-V	933	64	1832*	1842*	1830*	1835*	102	42	144	47
	Max-XP	480	312	1968*	2028*	1960*	1992*	228	150	311	193

Table 6.5 Computing time (seconds) including LSB inequalities selection time

Problem		1	2	3	4	5	6	7	8	9	10
No-C	No-LSB	10.21	34.23	n/a	n/a	n/a	n/a	69.52	383.80	20.49	127.12
	All-V	2.31	1.47	n/a	n/a	n/a	n/a	1.00	1.00	1.00	1.23
	Most-V	5.39	2.28	n/a	n/a	n/a	n/a	2.07	8.81	1.73	3.62
	Max-XP	5.02	4.44	n/a	n/a	n/a	n/a	6.95	32.01	3.61	13.96
C-Root	No-LSB	3.13	1.89	n/a	n/a	n/a	n/a	4.57	2.64	1.21	2.16
	All-V	1.00	1.70	n/a	n/a	n/a	n/a	2.82	3.32	1.75	1.06
	Most-V	5.16	2.23	n/a	n/a	n/a	n/a	3.91	8.87	1.77	3.38
	Max-XP	2.72	10.42	n/a	n/a	n/a	n/a	8.79	31.97	3.65	13.73
C-Def	No-LSB	3.11	1.42	n/a	n/a	n/a	n/a	4.62	3.90	1.10	2.08
	All-V	1.00	1.00	n/a	n/a	n/a	n/a	2.79	4.09	1.57	1.00
	Most-V	5.29	1.21	n/a	n/a	n/a	n/a	3.93	8.87	1.64	3.34
	Max-XP	2.72	5.93	n/a	n/a	n/a	n/a	8.81	31.97	3.54	13.65

Table 6.6 Normalized computing time including LSB inequalities selection time

Problem		1	2	3	4	5	6	7	8	9	10
No-C	No-LSB	4.09	2.58	4.31	15.65	6.31	3.36	2.47	1.55	4.10	3.49
	All-V	1.00	1.00	1.22	3.40	1.14	1.31	1.00	1.00	1.00	1.00
	Most-V	1.00	1.00	1.13	3.06	1.14	1.26	1.00	1.00	1.00	1.00
	Max-XP	1.00	1.00	1.08	3.06	1.14	1.34	1.00	1.00	1.00	1.00
C-Root	No-LSB	1.00	1.00	1.17	3.20	1.11	1.31	1.00	1.00	1.00	1.00
	All-V	1.00	1.00	1.22	3.12	1.04	1.19	1.00	1.00	1.00	1.00
	Most-V	1.00	1.00	1.12	3.20	1.04	1.30	1.00	1.00	1.00	1.00
	Max-XP	1.00	1.00	1.14	3.19	1.04	1.43	1.00	1.00	1.00	1.00
C-Def	No-LSB	1.00	1.00	1.18	3.28	1.17	1.31	1.00	1.00	1.00	1.00
	All-V	1.00	1.00	1.20	3.13	1.34	1.18	1.00	1.00	1.00	1.00
	Most-V	1.00	1.00	1.17	3.22	1.34	1.30	1.00	1.00	1.00	1.00
	Max-XP	1.00	1.00	1.14	3.42	1.34	1.38	1.00	1.00	1.00	1.00

Table 6.7 Optimality gap achieved (percent)

Problem		1	2	3	4	5	6	7	8	9	10
No-C	No-LSB	54,080	89,255	77,072	19,589	47,809	65,783	98,349	84,535	40,270	71,761
	All-V	53,912	88,955	77,141	19,402	47,380	65,390	97,942	84,448	40,228	71,715
	Most-V	53,969	88,896	77,088	19,381	47,380	65,371	97,942	84,478	40,267	71,715
	Max-XP	53,942	88,857	77,040	19,381	47,380	65,422	97,942	84,473	40,267	71,715
C-Root	No-LSB	53,920	88,963	77,096	19,391	47,331	65,395	98,106	84,585	40,269	71,781
	All-V	53,867	88,981	77,138	19,381	47,306	65,338	98,068	84,562	40,288	71,715
	Most-V	53,969	88,973	77,065	19,402	47,306	65,393	98,068	84,484	40,267	71,715
	Max-XP	53,867	89,065	77,089	19,402	47,306	65,465	98,068	84,473	40,267	71,715
C-Def	No-LSB	53,920	88,873	77,092	19,402	47,331	65,395	98,106	84,478	40,269	71,781
	All-V	53,867	88,930	77,108	19,381	47,407	65,338	98,068	84,478	40,283	71,715
	Most-V	53,969	88,889	77,088	19,402	47,407	64,565	98,068	84,484	40,267	71,715
	Max-XP	53,867	88,927	77,089	19,441	47,407	65,430	98,068	84,473	40,267	71,715

Table 6.8 Best objective found

Chapter 7

Solving the Capacitated Multi-Item Lot-Sizing Problem with Backorder and Integral Unit Production (CMBI) Using an MIP-Based Heuristic

7.1 Chapter Overview

In some manufacturing processes, products have to be produced in integral units. Once the process starts, it has to continue until it finishes without any interruption. Under the circumstance, it is not possible to finish part of the batch or the unit in this period, pause the process, and finish it in another period. Some examples of this kind of manufacturing process are obvious. In the foundry industry, for example, raw material is first melted. The melted metal is then treated with chemical, and poured or injected to the molds. The molds are later transported to the cooling area before they are separated from the part. Lastly, the part is cleaned and finished. During the melting process, the raw material requires a definite amount of time in the furnace before it is melted and ready for treatment. It is not possible to pause the process and resume it later on. Considering the transfer of the melting metal from the furnace to the mold, we fill up the molds with melting metal. Production plan has to be prescribed such that the melting metal is just enough to fill up the required number of molds. It is clear that in such system, production levels in a period have to be in integral units.

In this chapter, we focus on the more restricted version of CMB, which is presented in Chapter 6, where the integrality constraints are enforced on all of the production levels $x_{j,t}$'s. The problem is called CMBI. Our prime interest in this chapter is to develop a solution

method that gives *good* solutions within a reasonable amount of time, so that it can actually be used in practice. Moreover, in the case where optimality is more important than time, the solution method should be able to provide *optimal* solutions faster than available commercial solution methods.

The materials in this chapter are organized as follows. Section 7.2 re-introduces the mathematical formulation of CMBI. Section 7.3 gives a brief overview of the existing heuristics being used to solve complex problems. We developed the Cut-Relax-Fix-and-Free Heuristic (CRFF) to efficiently solve CMBI, and it is presented in Section 7.4. Computational results and discussion can be found in Section 7.5.

7.2 Mathematical Formulation

Following is the MIP formulation of CMBI. We use the same notation used in Chapter 6.

$$\text{(CMBI)} \quad Z(\Delta) = \min \sum_{j \in \mathcal{G}} \sum_{t \in \Omega} (f_j y_{gt} + h_j s_{gt} + b_j u_{gt}) \quad (7.1)$$

subject to

$$x_{gt} + s_{g(t-1)} + u_{gt} = d_{gt} + s_{gt} + u_{g(t-1)}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (7.2)$$

$$x_{gt} \leq \left(\frac{C_t}{p_g}\right) y_{gt}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (7.3)$$

$$\sum_{j \in \mathcal{G}} x_{gt} p_g \leq C_t \quad \forall t \in \Omega \quad (7.4)$$

$$\sum_{t \in \Omega} x_{gt} = \sum_{t \in \Omega} d_{gt}, \quad \forall g \in \mathcal{G} \quad (7.5)$$

$$y_{gt} \in \{0, 1\}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (7.6)$$

$$s_{gt}, u_{gt} \geq 0, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (7.7)$$

$$x_{gt} \in \{0, 1, 2, \dots\}, \quad \forall g \in \mathcal{G}, \forall t \in \Omega \quad (7.8)$$

As was done in Chapter 6, if we replace d_{gt} with \underline{d}_{gt} , the above formulation determines the cost for the committed demand, $Z(\underline{\Delta})$. Therefore, the cost of a tuple τ on supplier j ,

denoted by $c_j(\tau)$, can be defined by:

$$c_j(\tau) = Z(\mathbf{\Delta}) - Z(\underline{\mathbf{\Delta}}) \quad (7.9)$$

Recall from Part I that we assumed that each supplier j in \mathcal{N} has private knowledge about his costs on all possible tuples. Given that the supplier's cost determination model can be represented by CMBI, his cost on a tuple, $c_j(\tau)$, can be calculated by equation (7.9).

The objective function (7.1) is to minimize the total setup costs, inventory holding costs, and late penalty costs. Equation (7.2) represents inventory balance constraints. The setup state variable y_{gt} is controlled by the logical constraints (7.3). Inequality (7.4) limits the total production of a period. Constraints (7.5) ensure that, eventually, all demands will be fulfilled. Constraints (7.6) restrict the setup state variable y_{gt} to be binary, while constraints (7.7) force production, inventory and backloging levels to be nonnegative. Lastly, the non-negativity and integrality properties are imposed on all of the production levels x_{gt} 's in (7.8). We call the solution set satisfying (7.2) – (7.8) \mathcal{X}^{CMBI} .

In Section 6.2, we mentioned that CMB is \mathcal{NP} -hard. By having the integrality constraints for production level x_{gt} , CMBI is an even more restricted problem than CMB. Therefore, it can be inferred that CMBI is also \mathcal{NP} -hard.

7.3 Existing Heuristics for Complex Problems

The main objective of this chapter is to present a solution method that can give *good* solutions within a reasonable amount of time, so that the solution method can actually be used in practice. But, before we introduce our solution method, it is important to understand the strengths and weaknesses of the available heuristic algorithms that could be used to solve complex problems, including CMBI.

7.3.1 Local Search Heuristics

Local Search is a general technique used to find a good solution of a complex problem. If designed properly, the local search heuristic can possibly achieve very high-quality solutions. Basic elements of a local search heuristic are the *initial solution*, the definition of the *local neighborhood*, the *selection criterion* for the next initial solution, and the *termination criterion*. The general local search heuristic could be described as follows.

Step 1 Choose an initial solution

Step 2 Search for the solutions in the neighborhood

Step 3 Choose one or more new initial solutions

Step 4 Return to Step 1

Starting the heuristic procedure with a good initial solution is crucial to success. There is a good chance to achieve a very good, or even optimal solution, if the initial solution selected is near a very high quality solution. The question would then be how to define a neighborhood. Generally, it depends on the problem structure. For example, to find an optimal sequence of tasks a , b , and c with respect to some objective function, and given that an initial solution is the sequence (a, b, c) , neighborhoods of the initial solution could include, the pairwise interchanged solutions (b, a, c) and (a, c, b) . The design of the selection criterion used in Step 4 plays a crucial role in a local search heuristic. Another important issue that needs to be considered is the trade-off between the time spent in looking for solutions in the neighborhood and the quality of the solution required. Better solutions can be probably found, but of course, with the expense of more time.

There were tremendous efforts in developing heuristics that have the ability to escape from local optima with hope that it can achieve even better solutions. Examples of the well-known heuristics in this category include *Tabu Search*, *Simulated Annealing*, *Genetic Algorithms*. More details about these heuristic can be found in the references such as Morton and Pentico [75].

7.3.2 MIP-Based Heuristics

The heuristics mentioned in Section 7.3.1, with some customization to specific problems, could perform well in general case. However, they do not provide any guarantee about the optimality gap (formally defined later) they can achieve.

MIP-based heuristics, on the contrary, can provide information about the maximum discrepancy between the heuristic solution and optimal solution. The discrepancy is represented by the gap between the best integer solution and the best lower bound. Following are examples of general MIP-based heuristics.

7.3.2.1 Variable Rounding Heuristic

Suppose that we are considering an integer programming problem. Let \mathcal{Z} be a set of all integer numbers, $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ be an LP relaxation solution at any node in the branch-and-bound tree, and $\mathcal{F} = \{x_i^* : x_i^* \notin \mathcal{Z}\}$. This heuristic can be described as follows.

Step 1 Find $i = \arg \min_{i \in \mathcal{F}} \{x_i^* - \lfloor x_i^* \rfloor, \lceil x_i^* \rceil - x_i^*\}$

Step 2 If $x_i^* - \lfloor x_i^* \rfloor < 0.5$, set $x_i = \lfloor x_i^* \rfloor$, else set $x_i = \lceil x_i^* \rceil$

Step 3 Remove x_i from \mathcal{F}

Step 4 Solve this modified LP with x_i value fixed

Step 5 If the LP is infeasible, STOP (heuristic fails), else obtain new x^*

Step 6 If $\mathcal{F} \neq \emptyset$, go back to Step 1, else STOP (solution found)

Given an LP relaxation solution \mathbf{x}^* , the heuristic first determines the fractional variable x_i that is closest to an integer value. It then rounds (up or down) and fixes x_i to the closest integer. The modified LP is then solved. If it is found that the LP is infeasible, the heuristic stops, in which case it means that the heuristic fails. But if the LP is feasible, the heuristic proceeds to the next iteration. It repeats these iterations until the LP is found infeasible, or

all elements in \mathbf{x}^* are integer. This heuristic obviously can produce infeasibility even in the situation where feasible solutions do exist.

7.3.2.2 Relax-and-Fix Heuristic

Again, suppose that we consider an integer programming problem. Let $\mathcal{N} = \{1, \dots, n\}$, where n is the total number of integer variables, $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ be an LP relaxation solution, and $\mathcal{N}_1 \subset \mathcal{N}$. Here, we assume that, based on some criteria, we can determine that x_i , where $i \notin \mathcal{N}_1$, is more important than x_j , where $j \in \mathcal{N}_1$. The relax-and-fix heuristic can be described as follows.

Step 1 Relax the integrality constraints for all $x_i, i \in \mathcal{N}_1$

Step 2 Solve the resulting MIP and obtain \mathbf{x}^*

Step 3 Fix $x_i = x_i^*$ for all $i \notin \mathcal{N}_1$

Step 4 Solve the resulting IP

The relax-and-fix heuristic first relaxes the integrality constraints of the less important variables. Then it solves the resulting MIP. With the relaxation, the integer values of the more important variables are determined. Finally, these values are fixed, the integrality constraints of the less important variables are reinstated, and the resulting IP is solved. It is easy to see that by fixing some variables in Step 3, the feasible region of the problem is decreased in size. Consequently, the optimal solution of the problem in which some variables have been fixed can be suboptimal for the original problem.

7.4 Cut-Relax-Fix-and-Free Heuristic

This section presents the solution method that we develop for CMBI. As can be seen from Section 7.3.2.2, the Relax-and-Fix Heuristic intuitively makes sense because of the following two reasons. First, for an IP or an MIP, it is clear that solving a relaxed problem is easier than solving its original one. Second, fixing some of the variables in the original

problem using the relaxation's solution usually leads to near-optimal solutions with much less computing time. Our strategy is in part influenced by the Relax-and-Fix Heuristic.

However, by fixing some or all of the variables, the problem's solution set is made smaller. Consequently, the optimal solution obtained from the problem in which the variables have been fixed could be totally different from the optimal solution of the original problem. With this in mind, in addition to the intention to develop a solution method that provides good solution quickly, if optimality is really required, we intend to design a solution method that is capable of finding true optimal solution faster than of available branch-and-bound algorithms.

Suppose that we are considering an MIP problem \mathcal{P} . Let $\mathcal{N} = \{1, \dots, n\}$ be the index set for integer variables, and $\mathbf{x} = \{x_i : i \in \mathcal{N}\}$ be the set of integer variables. Also, we define new four sets $\emptyset = \mathcal{F}_A \subset \mathcal{F}_B \subset \mathcal{F}_C \subset \mathcal{F}_D = \mathcal{N}$. We denote by $\bar{\mathcal{P}}$ the relaxation of \mathcal{P} , $\mathbf{x}^* = \{x_i^* : i \in \mathcal{N}\}$ and $\bar{\mathbf{x}}^* = \{\bar{x}_i^* : i \in \mathcal{N}\}$ for optimal solutions of \mathcal{P} and $\bar{\mathcal{P}}$, respectively. Now, given positive integers δ_1 and δ_2 , consider the following four variable fixing schemes.

After solving the relaxation problem, add the following constraints:

Scheme A: $\lfloor \bar{x}_i^* \rfloor - \delta_1 \leq x_i \leq \lceil \bar{x}_i^* \rceil + \delta_2$, for all $i \in \mathcal{F}_A$

Scheme B: $\lfloor \bar{x}_i^* \rfloor - \delta_1 \leq x_i \leq \lceil \bar{x}_i^* \rceil + \delta_2$, for all $i \in \mathcal{F}_B$

Scheme C: $\lfloor \bar{x}_i^* \rfloor - \delta_1 \leq x_i \leq \lceil \bar{x}_i^* \rceil + \delta_2$, for all $i \in \mathcal{F}_C$

Scheme D: $\lfloor \bar{x}_i^* \rfloor - \delta_1 \leq x_i \leq \lceil \bar{x}_i^* \rceil + \delta_2$, for all $i \in \mathcal{F}_D$

Scheme A does not fix any of the integer variables ($\mathcal{F}_A = \emptyset$), while scheme D bounds all the integer variables ($\mathcal{F}_D = \mathcal{N}$). We denote by \mathcal{X}_i , z_i , and z_i^* the solution set, the incumbent value at any given time, and the optimal value of \mathcal{P} when fixing scheme i is applied, respectively. Since $\mathcal{F}_A \subset \mathcal{F}_B \subset \mathcal{F}_C \subset \mathcal{F}_D$, it implies that $\mathcal{X}_D \subset \mathcal{X}_C \subset \mathcal{X}_B \subset \mathcal{X}_A$ and also $z_D^* \geq z_C^* \geq z_B^* \geq z_A^*$ (for minimization problems). However, in terms of the solution time from the starting solution to the optimal solution, the problems with larger solution sets tend to require more time than the problems with smaller solution sets. In other words, if t_i is the

time used to solve \mathcal{P} with fixing scheme i , we can generally expect that $t_D < t_C < t_B < t_A$. Regarding the starting solution, it is natural to hypothesize that larger the solution set worse the starting solution. We will substantiate this hypothesis by looking at fixing schemes A and D. By solving a relaxation problem, even though $\bar{\mathbf{x}}_D^*$ is probably infeasible for \mathcal{X}_D , by fixing the variables, the starting objective is likely to be not too far from z_D^* , especially in the cases where δ_1 and δ_2 are very small.

In Figure 7.1, we illustrate our hypothesis on the change of the incumbent value, z_i , over time when problem \mathcal{P} is solved using the four fixing schemes. There, we assume that we spend t_D time units to solve $\bar{\mathcal{P}}$. It can be seen that, by fixing more variables, the starting objective is lower. But over time, the problem with fewer variables fixed will reach a better solution. Hence, if we fix some of the variables in the beginning stage and remove the variable fixing constraints in later stage, we are likely to get high-quality solutions in the beginning without giving up the optimal solution in the end. Using this idea, together with the valid inequality selection procedure introduced in Chapter 6, we developed the Cut-Relax-Fix-and-Free Heuristic (CRFF) to solve the CMBI problem.

Let $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}, \bar{\mathbf{u}}) = (x_{1,1}^-, \dots, x_{gt}^-, y_{1,1}^-, \dots, y_{gt}^-, s_{1,1}^-, \dots, s_{gt}^-, u_{1,1}^-, \dots, u_{gt}^-)$ be a feasible solution for CMB. We now formally introduce the Cut-Relax-Fix-and-Free Heuristic.

Cut-Relax-Fix-and-Free Heuristic:

Step 1 Select and add valid inequalities into CMB

Step 2 Solve CMB for t_1 seconds and obtain $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}, \bar{\mathbf{u}})$

Step 3 Convert CMB to CMBI and set $x_{gt} \geq \lfloor x_{gt}^- \rfloor$, for all $1 \leq t \leq T_1$

Step 4 Solve the resulting CMBI for another t_2 seconds, and obtain objective z_c

Step 5 Remove the constraints added in Step 3, and set the cut off value (upper bound) at z_c

Step 6 Solve the resulting CMBI for another t_3 seconds

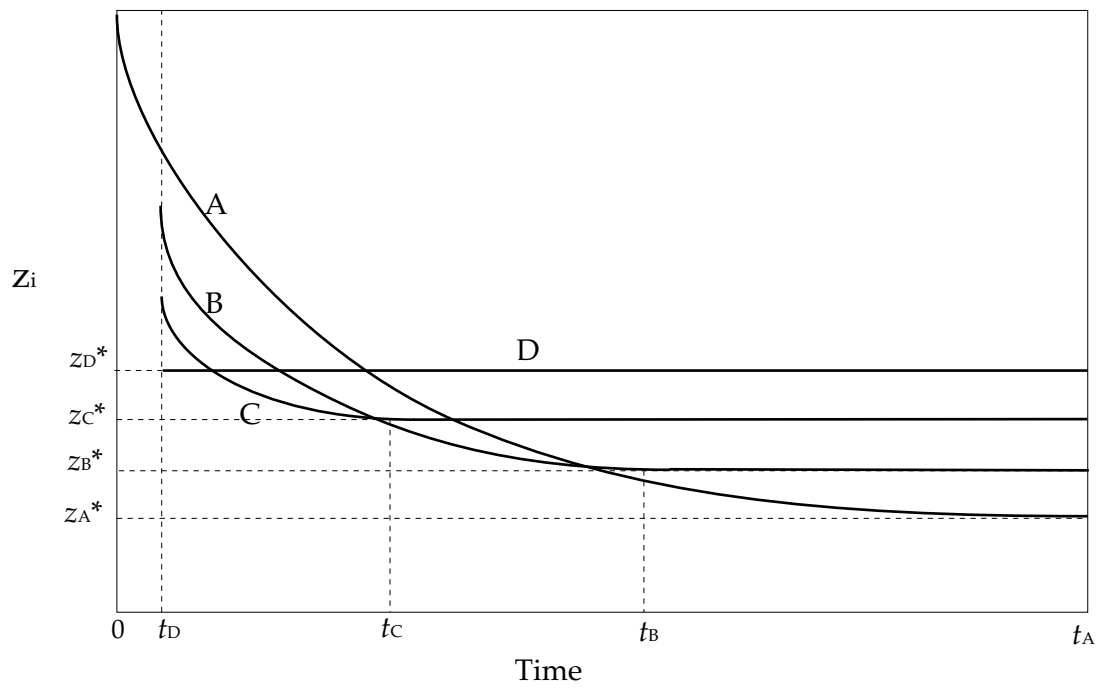


Figure 7.1 Hypothetical z_i change over time

Four parameters that we need to assign the values before running the solution method are t_1 , t_2 , t_3 , and T_1 . The appropriate values for those parameters can be set depending upon the application CRFF is being used for. Also, in Step 1, we need to decide the valid inequalities to be added. This could be taken care of by any of the valid inequality selection procedures presented in Chapter 6. The relaxed problem (CMB) is solved in Step 2. At this point, the trade-off between time and quality of the CMB solution have to be made. But generally, any close-to-optimal solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}, \bar{\mathbf{u}})$ would be satisfactory. In Step 3, because we set $x_{gt} \geq \lfloor \bar{x}_{gt} \rfloor$, the setup variable y_{gt} may be affected through constraint (7.3). Note also, by setting the lower bound of x_{gt} at the floor value of \bar{x}_{gt} , the feasible solution will definitely be found if CMBI is in fact feasible. All of the constraints added in Step 3 are removed in Step 5. This enables the solution method to eventually achieve the true optimal solution if t_3 is sufficiently large. Also in Step 5, we set the cut off value at z_2 . In doing so, all the nodes with higher LP objectives will be fathomed, and consequently, fewer number of nodes will need to be processed. In the next section, we will examine the performance of CRFF and compare it with other solving schemes.

7.5 Implementations and Computational Results

It is desirable, of course, to have a solution algorithm that solves problems to optimality quickly. But most practical problems are complex and it is impractical to solve them to optimality. Normally, solution quality, or optimality for that matter, has to be traded off with the required computing time. Our goal in this section is to find out if our CRFF heuristic really proves to be efficient in terms of being able to return good solutions, and also has a good chance of reaching optimal solution faster than commercially available optimization software.

7.5.1 Solving Strategies Considered

We compare an implementation of Cut-Relax-Fix-and-Free Heuristic introduced in Section 7.4 with three solving strategies implemented by a commercial optimizer. In order to

evaluate the four solving strategies, the objective values of the four strategies over time will be plotted and compared. Each strategy has an equal run time limit of 300 seconds.

CRFF(10,140,150) The results from Section 6.5.3 indicate that the solution strategy No-C/All-V is the fastest in achieving one percent optimality gap. As a result, we do not add any CPLEX cuts throughout the CRFF process and we implement Step 1 of CRFF (cut selection process) by employing All-V strategy. The numbers 10, 140, and 150 represent the t_1 , t_2 , and t_3 (See CRFF description). In the experiment, we set $T_1 = 10$.

C-Root(300) This strategy adds CPLEX cuts at the root node, then unless it finds optimal solution sooner, it solves the problem for 300 seconds. There will be no LSB inequalities added during the process.

C-Def(300) This is a CPLEX's default solving strategy for any IP or MIP problems. CPLEX cuts are added at any node in the branch-and-bound tree the solver deems appropriate. Unless the optimal solution is found, the strategy is allowed to run for 300 seconds before being terminated. There will be no LSB inequalities added during the process.

No-Cuts(300) During the solving process, the strategy simply solves the original MIP formulation without adding any LSB or CPLEX cuts. The running stops if 300 seconds is reached or before that if the optimal solution is found.

7.5.2 Test Problems

Ten of the test problems used in Chapter 6 are modified such that all of the production levels x_{gt} 's are restricted to be non-negative integers. These ten modified problems are used as test problems in this chapter.

7.5.3 Results

Each of the ten test problems is solved with the four strategies described above. At every ten seconds, we record the objective value. A graph showing objectives of all the strategies over time can be produced. While the graph could be useful in tracing the objectives of different strategies, it does not provide information regarding the optimality gap an objective achieved.

Let \bar{z}_{300} be the highest lower bound of the four strategies found after they have been run for 300 seconds. We compute the *terminal optimality gap*, denoted by ω , by

$$\omega = \frac{z - \bar{z}_{300}}{\bar{z}_{300}} \times 100 \quad (7.10)$$

Obviously, ω is an optimality gap between z and \bar{z}_{300} .

In Figure 7.2– 7.6, we plot ω of the four strategies against time elapsed. At ten seconds, CRFF finished solving CMB and just starts solving CMBI. It is clear that, starting from 20 seconds until 300 seconds, CRFF(10,140,150) almost always achieves a lower optimality gap than other strategies. The only two exceptions are in Problem 4 and 6, in which cases No-Cuts(300) and C-Root(300) achieve a marginally better optimality gap than CRFF(10,140,150), i.e., 7.55 vs. 7.67 percent and 4.43 vs. 4.87 percent, respectively. While CRFF(10,140,150) performs consistently well, other three strategies can occasionally perform very badly. For example, in Problem 6, the ω values of C-Root(300) and C-Def(300) remain at the levels around 28 percent for 280 seconds, while during the same period CRFF(10,140,150)'s ω values stay just above five percent. An important and rather surprising observation is that No-Cuts(300), in many instances, even outperforms C-Root(300) and C-Def(300). This could be an indication that, as opposed to LSB inequalities, CPLEX cuts are not usually helpful in solving CMBI.

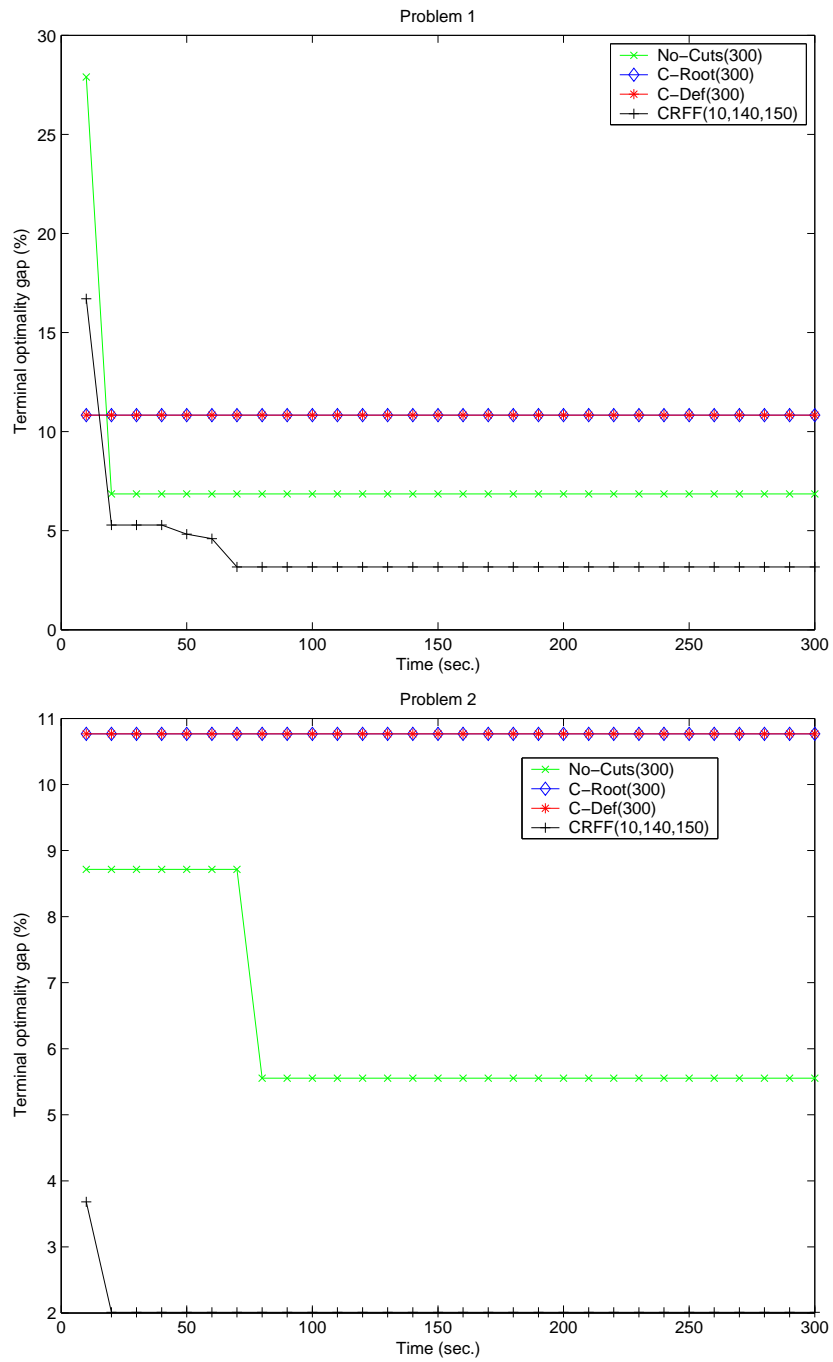


Figure 7.2 The change of terminal optimality gap over time for Problems 1 and 2

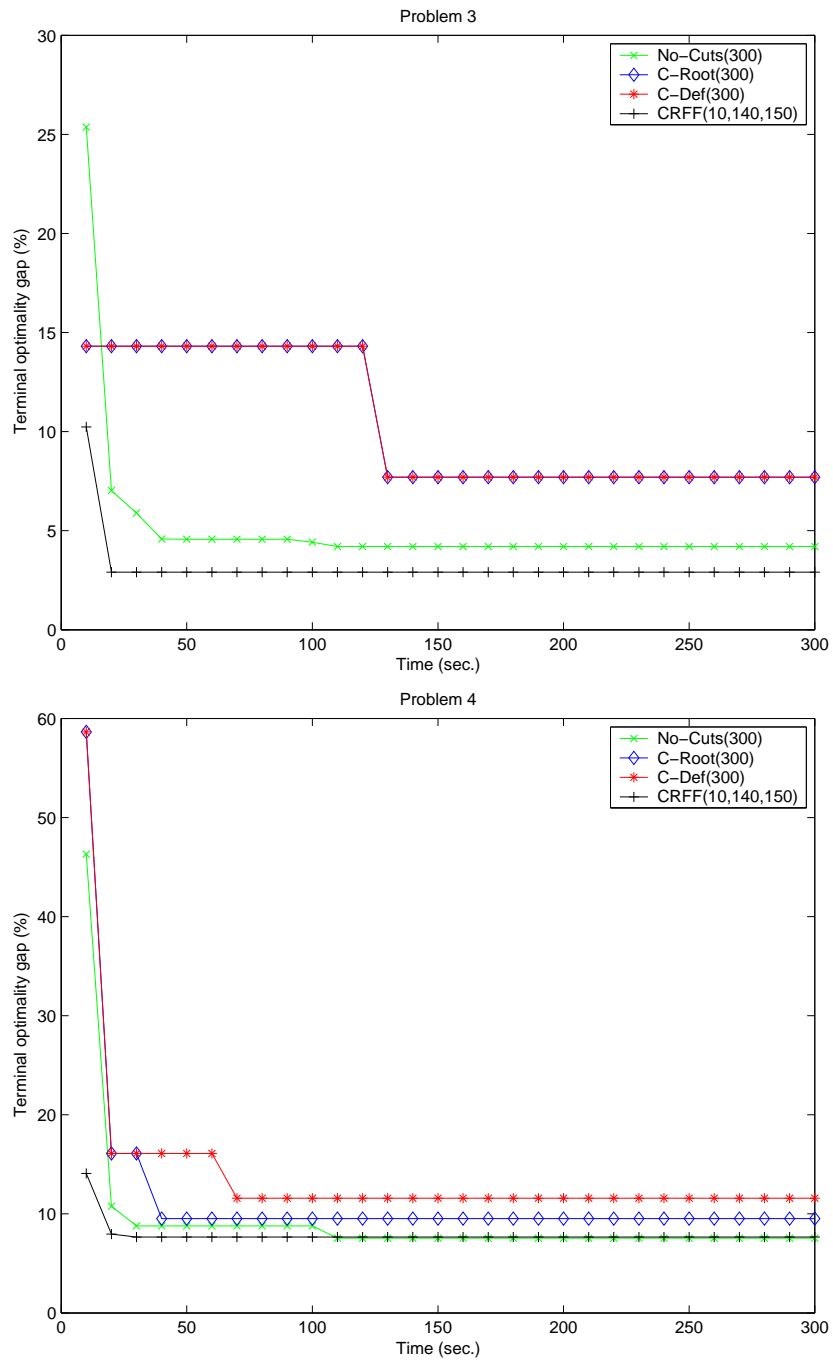


Figure 7.3 The change of terminal optimality gap over time for Problems 3 and 4

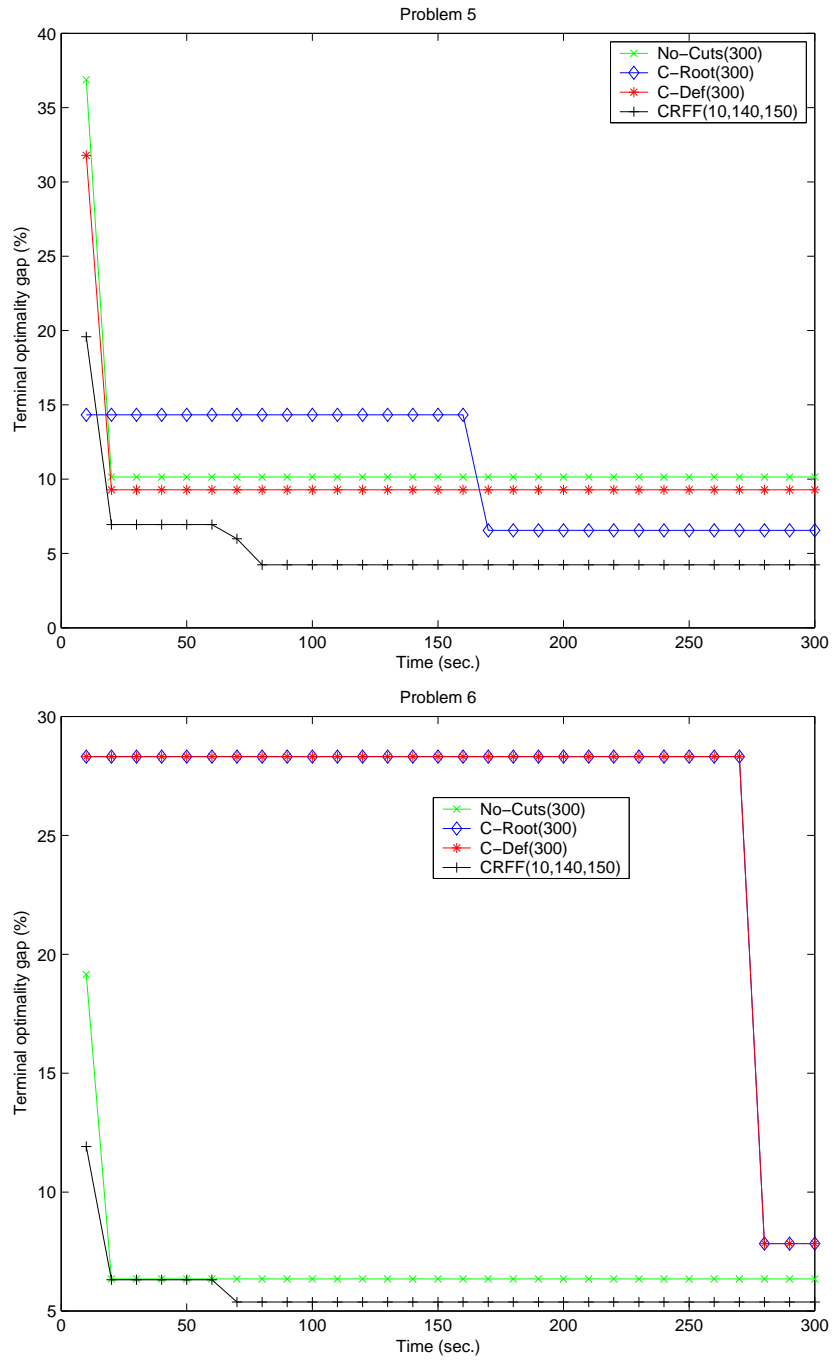


Figure 7.4 The change of terminal optimality gap over time for Problems 5 and 6

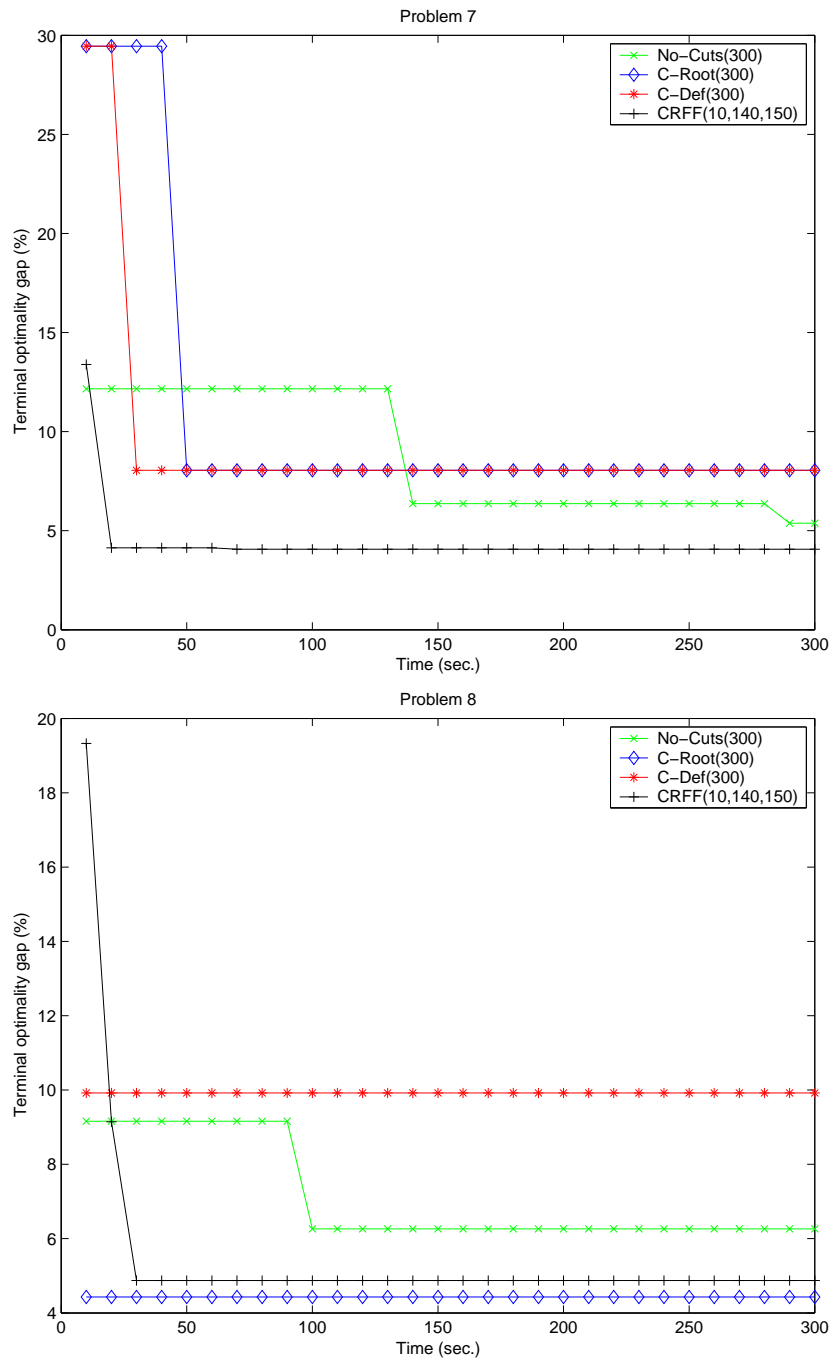


Figure 7.5 The change of terminal optimality gap over time for Problems 7 and 8

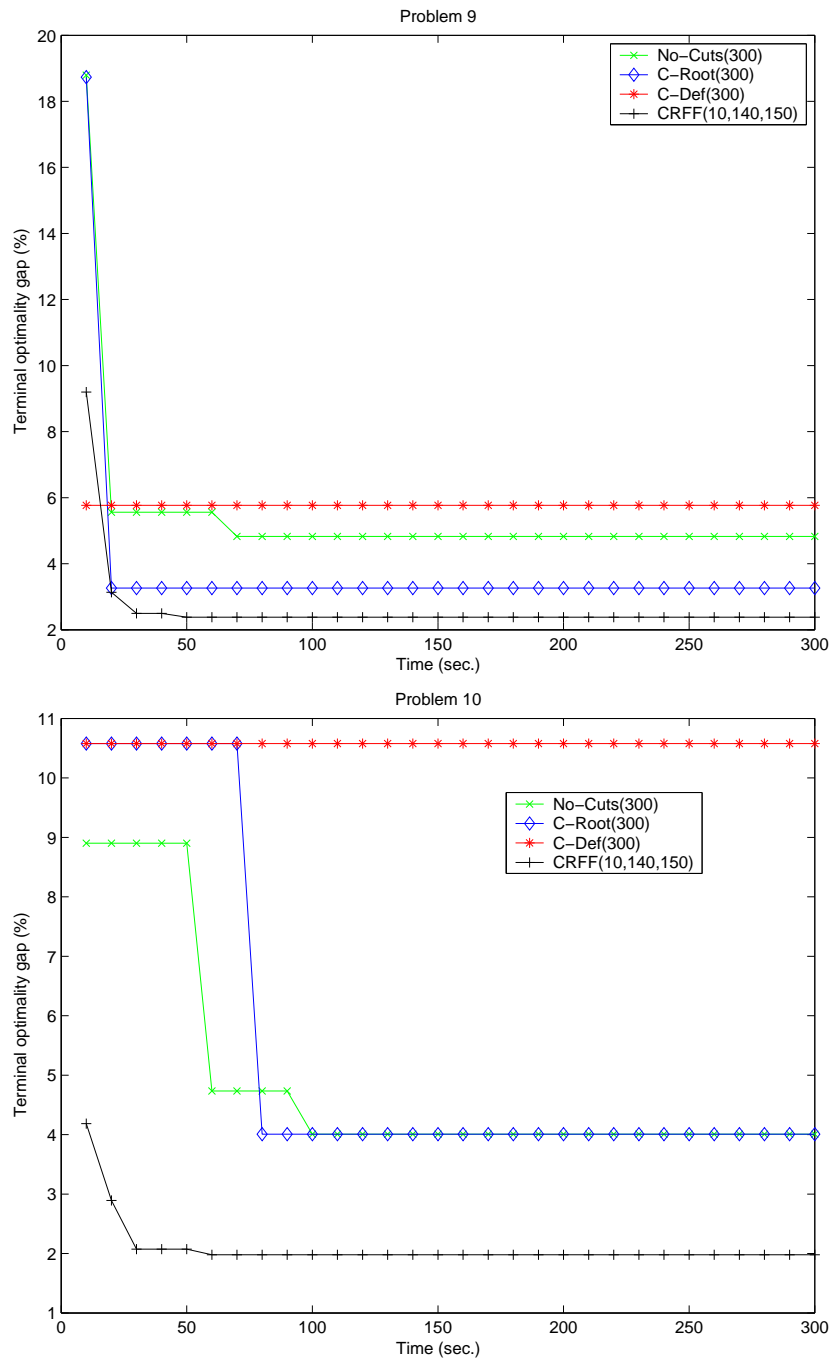


Figure 7.6 The change of terminal optimality gap over time for Problems 9 and 10

Part IV

Future Research

Chapter 8

Future Research

8.1 Chapter Overview

This chapter presents promising future research directions involving outsourcing mechanisms and cost determination algorithms for suppliers.

8.2 Outsourcing Mechanisms

There remain a large number of research issues dealing with outsourcing mechanisms that can be explored. As far as we are concerned, we can immediately extend our work to address the following interesting issues.

Multi-Seller Model

Bikhchandani [16] proves that, for exchange economy, there always exists third-order pricing equilibrium for the single-seller model. In this research, we also show that a competitive equilibrium (analogous to the third-order pricing equilibrium defined by Bikhchandani [16]) always exists in our single-seller outsourcing economy. An interesting area of future research would be to extend the outsourcing model to cover the multi-seller case and analyze whether the multi-seller model provides any pricing equilibrium.

Algorithm's Payments vs. Vickrey Payments

Leonard [59] studies a direct mechanism for the assignment problem of individual-position matchings. He shows that at the minimum price equilibrium each individual makes the

Vickrey payment and receives his marginal product. Therefore, there is no incentive for individuals to misrepresent their preferences. A similar result is obtained in Bikhchandani [16] for the single-seller exchange economy with multi-unit demand buyers.

We show in Chapter 4 that `OUTSOURCING` terminates in a competitive equilibrium. However, we have yet to prove that the achieved equilibrium prices are Vickrey payments. To do that, we first have to show that there exists a unique minimum competitive equilibrium in the outsourcing economy. Then, we have to examine if the competitive equilibrium achieved is (approximately) the minimum one.

8.3 Improvements in Cost Determination Algorithms

Many practical problems call for MIP models. For many years, researchers have put tremendous efforts into developing efficient solution techniques for MIP's. However, for most practical problems, the computing times are still too long, even when solved with the latest techniques. The research opportunities in this area are virtually limitless. Following are some of the interesting future research topics related to our work.

New Families of Valid Inequalities

New families of the valid inequalities derived specifically for CMBI can be found. It is possible that these valid inequalities can prove to be more efficient than the LSB inequalities.

Algorithmic Improvement in Most-V and Max-X Strategies

The author believes that, with some technical adjustments, Most-V and Max-X strategies can become more efficient in solving both CMB and CMBI. As described in Chapter 6, the major drawback of the strategies is the time required in the input/output process. More efficient data storing/data retrieving techniques or more appropriate data structure can make these strategies more attractive.

Applying LSB Inequalities Using Branch-and-Cut Algorithm

In this research, we apply the LSB inequalities using the Cut-and-Branch strategy. Branch-and-Cut is another widely used algorithm, where cuts can be added not only at the root node, but also at any other nodes in the tree. Even though there is no guarantee that applying the LSB cuts using Branch-and-Cut Algorithm will improve the solution time, it is probably worthwhile to examine the possibility.

New Valid Inequality Selection Criteria

In this research, we select a valid inequality based on the calculated violation magnitude. This magnitude may or may not be the best criteria in selecting valid inequalities. Further investigation on the valid inequality selection criteria could result in more efficient solution algorithms.

Investigation of Parameter Selection for CRFF

In order to make CRFF operate at its optimal performance, we need to investigate the appropriate levels of the parameters used in the heuristic. Extensive experiments may have to be conducted in the investigation.

LIST OF REFERENCES

- [1] E-sourcing: Negotiating value in a volatile economy. White paper, Aberdeen Group, Inc., Boston, April 2001.
- [2] P. Afentakis and B. Gavish. Optimal lot sizing algorithms for complex product structures. *Operations Research*, 34:237–249, 1986.
- [3] A. Aggarwal and J. K. Park. Improved algorithms for economic lot-size problems. *Operations Research*, 41(3):549–571, 1993.
- [4] Agostinho Agra and Miguel Constantino. Lotsizing with backlogging and start-ups: the case of wagner-whitin costs. *Operations Research Letters*, 25:81–88, 1999.
- [5] E. Arkin, D. Joneja, and R. Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8:61–66, 1989.
- [6] H. C. Bahl, L. P. Ritzman, and J. N. D. Gupta. Deterministic lot sizes and resource requirements: A review. *Operations Research*, 35(3):329–345, 1987.
- [7] Keneth R. Baker. An experimental study of the effectiveness of rolling schedules in production planning. *Decision Science*, 8:19–27, 1977.
- [8] Imre Barany, Tony J. van Roy, and Laurence A. Wolsey. Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10):1255–1261, 1984.
- [9] James C. Bean and Robert L. Smith. Conditions for the existence of planning horizons. *Mathematics of Operations Research*, 9(3):391–401, 1984.
- [10] G. Belvaux and L.A. Wolsey. *bc-prod*: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46:724–738, 2000.
- [11] D. P. Bertsekas. A distributed algorithm for the assignment problem. Working paper, Laboratory for Information and Decision Systems, M.I.T., March 1979.
- [12] D. P. Bertsekas. A distributed asynchronous relaxation algorithm for the assignment problem. In *Proc. 24th IEEE Conf. Decision and Control*, pages 1703–1704, Dec. 1985.

- [13] D. P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14:105–123, 1988.
- [14] C. Bes and S. P. Sethi. concepts of forecast and decision horizons: Applications to dynamic stochastic optimization problems. *Mathematics of Operations Research*, 13(2):295–310, 1988.
- [15] Sushil Bikhchandani, Sven de Vries, James Schummer, and Rakesh V. Vohra. Linear programming and vickrey auctions. volume 127, *Mathematics of the Internet of IMA Volumes in Mathematics and its Applications*, pages 75–116. Springer, 2001. Appeared in: <http://www.springer-ny.com/detail.tpl?ISBN=0387953590>.
- [16] Sushil Bikhchandani and Joseph M. Ostroy. The package assignment model. Technical report, Anderson School of Management, University of California, Los Angeles, June 2001.
- [17] P. J. Billington, J. O. McClain, and L. J. Thomas. Heuristics for multilevel lot-sizing with a bottleneck. *Management Science*, 32(8):989–1006, 1986.
- [18] Peter J. Billington, John O. McClain, and Joseph Thomas. Mathematical programming approaches to capacity-constrained mrp systems: Review, formulation and problem reduction. *Management Science*, 29(10):1126–1141, 1983.
- [19] Gabriel R. Bitran and Horacio H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28:1174–1186, 1982.
- [20] Joseph D. Blackburn and Howard Kunreuther. Planning horizons for the dynamic lot size model with backlogging. *Management Science*, 21(3):251–255, 1974.
- [21] Jeremy Bulow and Paul Klemperer. Auctions vs. negotiations. *American Economic Review*, 86:180–194, 1996.
- [22] D. Cattrysse, M. Salomon, R. Kuik, and L. N. van Wassenhove. A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup-times. *Management Science*, 39:477–486, 1993.
- [23] Suresh Chand and Thomas E. Morton. Minimal forecast horizon procedures for dynamic lot size models. *Naval Research Logistics*, 33:111–122, 1986.
- [24] Suresh Chand, Suresh P. Sethi, and Jean-Marie Proth. Existence of forecast horizons in undiscounted discrete-time lot size models. *Operations Research*, 38(5):884–892, 1990.
- [25] Suresh Chand, Suresh P. Sethi, and Gerhard Sorger. Forecast horizons in the discounted dynamic lot size model. *Management Science*, 38(7):1034–1048, 1991.

- [26] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.
- [27] Miguel Constantino. A cutting plane approach to capacitated lot-sizing with start-up costs. *Mathematical Programming*, 75:353–376, 1996.
- [28] Gabrielle Demange, David Gale, and Marilda Sotomayor. Multi-item auctions. *Journal of Political Economy*, 94(4):863–872, 1986.
- [29] P. Dixon and E. A. Silver. A heuristic solution procedure for the multi-item single-level, limited capacity, lot-sizing problem. *Journal of Operations Management*, 2(1):23–39, 1981.
- [30] Gary D. Eppen and R. Kipp Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.
- [31] Awi Federgruen and Michal Tzur. A simple forward algorithm to solve general dynamic lot sizing models with n periods in $o(n \log n)$ or $o(n)$ time. *Management Science*, 37(8):909–925, 1991.
- [32] Awi Federgruen and Michal Tzur. The dynamic lot-sizing model with backlogging: A simple $o(n \log n)$ algorithm and minimum forecast horizon procedure. *Naval Research Logistics*, 40:459–478, 1993.
- [33] Awi Federgruen and Michal Tzur. Minimal forecast horizons and a new planning procedure for the general dynamic lot sizing model: Nervousness revisited. *Operations Research*, 42:456–468, 1994.
- [34] Bernhard Fleischmann. Discrete lot-sizing and scheduling problem. *European Journal of Operational Research*, 44(3):337–348, 1990.
- [35] Bernhard Fleischmann. The discrete lot-sizing and scheduling problem with sequence-dependent setup costs. *European Journal of Operational Research*, 75:395–404, 1994.
- [36] M. Florian, J. K. Lenstra, and A. H. G. Rinnoy Kan. Deterministic production planning: Algorithms and complexity. *Management Science*, 26:669–679, 1980.
- [37] Micheal Florian and Morton Klien. Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18(1):12–20, 1971.
- [38] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, Cambridge, Massachusetts, 1996.
- [39] R. E. Gomory. An algorithm for the mixed integer problem. Research Memorandum RM-2597, The RAND Corporation, 1960.
- [40] Theodore Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

- [41] Faruk Gul and Ennio Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87(1):95–124, 1999.
- [42] S. T. Hackman and R. C. Leachman. A general framework for modelling production. *Management Science*, 35(4):478–495, 1989.
- [43] F. W. Harris. How many parts to make at once. *Factory, The Magazine of Management*, 10(2):135–136, 1913.
- [44] K. S. Hindi. Efficient solution of the single-item, capacitated lot-sizing problem with start-up and reservation costs. *Journal of the Operational Research Society*, 46(10):1223–1236, 1995.
- [45] K. S. Hindi. Solving the clsp by a tabu search heuristic. *Journal of the Operational Research Society*, 47(1):151–161, 1996.
- [46] Vernon Ning Hsu. Dynamic economic lot size model with perishable inventory. *Management Science*, 46(8):1159–1169, 2000.
- [47] James P. Ignizio and Tom M. Cavalier. *Linear Programming*. Prentice-Hall, Inc., New Jersey, 1994.
- [48] U. Karmarkar, S. Kekre, and S. Kekre. The dynamic lot-sizing problem with startup and reservation costs. *Operations Research*, 35:389–398, 1987.
- [49] U. Karmarkar and L. Schrage. The deterministic dynamic product cycling problem. *Operations Research*, 33:326–345, 1985.
- [50] Alexander S. Kelso and Vincent P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50(6):1483–1504, 1982.
- [51] A. Kimms. Competitive methods for multi-level lot sizing and scheduling: Tabu search and randomized regrets. *International Journal of Production Research*, 34(8):2279–2298, 1996.
- [52] A. Kimms. A genetic algorithm for multi-level, multi-machine lot sizing and scheduling. *Computers and Operations Research*, 26(8):829–848, 1999.
- [53] Paul Klemperer. What really matters in auction design. Working paper, University of Oxford, August 2001.
- [54] Vijay Krishna. *Auction Theory*. Academic Press, San Diego, 2002.
- [55] Roelof Kuik and Marc Salomon. Batching decisions: Structure and models. *European Journal of Operational Research*, 75:243–263, 1994.

- [56] Roelof Kuik, Marc Salomon, Luk N. van Wassenhove, and Johan Maes. Linear programming, simulated annealing and tabu search heuristics for lotsizing in bottleneck assembly systems. *IIE Transactions*, 25(1):62–71, 1993.
- [57] L. S. Lasdon and R. C. Terjung. An algorithm for multi-item scheduling. *Operations Research*, 19:946–969, 1971.
- [58] Chung-Yee Lee, Sila Cetinkaya, and Albert P. M. Wagelmans. A dynamic lot-sizing model with demand time windows. Technical report, Econometric Institute Report EI-9948/A, 1999.
- [59] Herman B. Leonard. Elicitation of honest preferences for the assignment of individuals to positions. *Journal of Political Economy*, 91:461–479, 1983.
- [60] J. Leung, T. Magnanti, and R. Vachani. Facets and algorithms for capacitated lot-sizing. *Mathematical Programming*, 45:331–359, 1989.
- [61] Vahid Lotfi and Yong Seok Yoon. Algorithm for the single-item capacitated lot-sizing problem with concave production and holding costs. *Journal of the Operational Research Society*, 45(8):934–941, 1994.
- [62] Sebastian Lozano, Juan Larraneta, and Luis Onieva. Primal-dual approach to the single level capacitated lot-sizing problem. *European Journal of Operational Research*, 51(3):354–366, 1991.
- [63] J. Maes, J. O. McClain, and L. N. Wassenhove. Multilevel capacitated lotsizing complexity and lp-based heuristics. *European Journal of Operational Research*, 53:131–148, 1991.
- [64] J. Maes and L. N. van Wassenhove. Multi-item single-level capacitated dynamic lot-sizing heuristic: A computational comparison (part i: Static case). *IIE Transactions*, 18:114–123, 1986.
- [65] J. Maes and L. N. van Wassenhove. Multi-item single-level capacitated lot-sizing heuristics: A general review. *Journal of the Operational Research Society*, 39(11):991–1004, 1988.
- [66] Timothy M. Magee and Fred Glover. Integer programming. volume 20 of *Industrial Engineering*, chapter 3, pages 123–269. Marcel Dekker, Inc., 1996.
- [67] T. Magnanti and R. Vachani. A strong cutting plane algorithm for production scheduling with changeover costs. *Operations Research*, 38:456–473, 1990.
- [68] R. Kipp Martin. Generating alternative mixed-integer programming models using variable redefinition. *Operations Research*, 35(6):820–831, 1987.

- [69] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, Oxford, 1995.
- [70] R. Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25:699–738, 1987.
- [71] R. Preston McAfee and John McMillan. Game theory and competition. *Journal of Marketing Research*, 33:263–267, 1996.
- [72] Andrea Meyer. Logistics on the internet: Auctions and fulfillment. Newsletter and calendar, MIT Center for Transportation and Logistics, 2000. <http://web.mit.edu/ctl/www/news/50/internetlogistics.htm>.
- [73] Harvey H. Millar and Minzhu Yang. Lagrangian heuristics for the capacitated multi-item lot-sizing problem with backordering. *International Journal of Production Economics*, 34(1):1–15, 1994.
- [74] Thomas E. Morton. An improved algorithm for the stationary cost dynamic lot size model with backlogging. *Management Science*, 24(8):869–873, 1978.
- [75] Thomas E. Morton and David W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley & Sons, Inc., 1993.
- [76] S. Nahmias. Perishable inventory theory: A review. *Operations Research*, 30:680–708, 1982.
- [77] George L. Nemhauser and Laurence A. Wolsey. Integer programming. volume 1 of *Handbooks in Operations Research and Management Science*, chapter 6, pages 447–527. Elsevier Science Publishers B. V., Amsterdam, 1989.
- [78] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1999.
- [79] David C. Parkes. Iterative combinatorial auctions: Achieving economic and computational efficiency. Dissertation, University of Pennsylvania, 2001.
- [80] Y. Pochet and L. Wolsey. Solving multi-item lot-sizing problems using strong cutting planes. *Management Science*, 37:53–67, 1991.
- [81] Yves Pochet and Laurence A. Wolsey. Algorithms and reformulations for lot sizing problems. volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter 6, pages 245–293. American Mathematical Society, 1995.
- [82] C. N. Potts and L. N. van Wassenhove. Integrating scheduling with batching and lotsizing: A review of algorithms and complexity. *Journal of Operational Research Society*, 43(5):395–406, 1992.

- [83] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, Massachusetts, 1998.
- [84] H. M. Salkin and K. Mathur. *Foundations of Integer Programming*. North-Holland, New York, 1989.
- [85] M. Salomon. Deterministic lotsizing models for production planning. volume 355 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1991.
- [86] M. Salomon, M. M. Solomon, L. N. van Wassenhove, Y. D. Dumas, and S. Dauzere-Peres. Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the traveling salesman problem with time windows. *European Journal of Operational Research*, 100, 1997.
- [87] Lloyd S. Shapley and Martin Shubik. The assignment game i: The core. *International Journal of Game Theory*, 1(2):111–130, 1972.
- [88] Dong X. Shaw and Albert P. M. Wagelmans. Algorithm for single-item capacitated economic lot sizing with piecewise linear production costs and general holding costs. *Management Science*, 44(6):831–838, 1998.
- [89] H. A. Taha. *Integer Programming: Theory, Applications, and Computations*. Academic Press, New York, 1975.
- [90] J. M. Thizy and L. N. van Wassenhove. Lagrangean relaxation for the multi-item capacitated lot-sizing problem: A heuristic implementation. *IIE Transaction*, March:308–313, 1985.
- [91] William W. Trigeiro. A dual-cost heuristic for the capacitated lot sizing problem. *IIE Transactions*, March:67–72, 1987.
- [92] C. P. M. van Hoesel and A. P. M. Wagelmans. An $o(t^3)$ algorithm for the economic lot-sizing problem with constant capacity. *Management Science*, 42(1):142–150, 1996.
- [93] Hal R. Varian. Mechanism design for computerized agents. In *Usenix Workshop on Electronic Commerce*, 1995.
- [94] Hal R. Varian. *Intermediate Microeconomic: A Modern Approach*. W. W. Norton & Company, New York, 1999.
- [95] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.
- [96] Albert Wagelmans and Stan van Hoesel. Economic lot sizing: An $o(n \log n)$ algorithm that runs in linear time in the wagner-within case. *Operations Research*, 40:S145–S156, 1992.

- [97] Harvey M. Wagner and Thomas M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 13(1):89–96, 1958.
- [98] Joachim P. Walser, Ramesh Iyer, and Narayanan Venkatasubramanian. Integer local search method with application to capacitated production planning. In *Proceedings of the 15th National Conference on Artificial Intelligence, Madison, WI, USA*, pages 373–379. AAAI, 1998.
- [99] Robert B. Wilson. Strategic analysis of auctions. volume 1 of *Handbook of Game Theory with Economic Applications*, pages 227–279. ElsevierScience Publishers, Amsterdam, 1992.
- [100] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.
- [101] Willard I. Zangwill. A deterministic multi-period production scheduling model with backlogging. *Management Science*, 13:105–119, 1966.