Construction of Cost-Delivery Date Frontier for Capacitated Multi-Item Lot-Sizing System with Backorders

Seeronk Prichanont (prichano@cae.wisc.edu)
Department of Industrial Engineering, University of Wisconsin-Madison

Dharmaraj Veeramani (raj@ie.engr.wisc.edu) *
Department of Industrial Engineering, University of Wisconsin-Madison
266B Mechanical Engineering Building,
1513 University Avenue, Madison, WI 53706-1572, USA.
Tel: +1-608-262-0861; Fax: +1-608-262-8454

Keywords: Cost-delivery date frontier; price-delivery date bid frontier; capacitated multi-item lot-sizing system; valid inequality

*Corresponding author

Construction of Cost-Delivery Date Frontier for Capacitated Multi-Item Lot-Sizing System with Backorders

Abstract

A new sourcing process in which suppliers are allowed to submit the Price-Delivery Date Bid Frontier (PDDF), containing multiple price-delivery date options, is considered. In order to construct the PDDF, based on the current shop-floor information, such as committed demand and cost information, supplier first determines the Cost-Delivery Date Frontier (CDDF) containing minimum cost-delivery date tuples. This paper addresses the problem of how to quickly and accurately develop a cost-delivery date tuple. Assuming that the supplier's shop-floor can be represented by a capacitated multi-item lot-sizing model with backorders, a family of valid inequalities, called LSB inequalities, and three valid inequality selection strategies are proposed. Test problems are formulated in GAMS modeling language and solved by CPLEX solver, by which some generic cuts can be generated. Computational results show that a cost-delivery date tuple can be determined quickest by adding only the LSB inequalities through a valid inequality selection strategy called All-V.

1 Introduction

In the sourcing process, price is not the only attribute that customers consider. For manufacturing companies, for example, in addition to price, delivery lead time is another important consideration in supplier selection. Improper delivery lead time could have impacts on cost (e.g. inventory cost or late penalty cost) and eventually profit. In this research, as a consequence, delivery date is considered as the second attribute in decision making to select a supplier.

Most of the sourcing decisions, in practice, involve two decision making stages. First stage is for the sourcing company to select a set of supplier compa-

nies. The selection could be based upon price agreement for a particular period of time or upon the previous relationship between those companies. In the second stage, when the sourcing company faces a particular demand, based on the known price list, it will make decision on the delivery date and the suppliers to whom the order will be granted.

There are drawbacks for this two-stage sourcing decision process, however. The process requires customer and suppliers to make contractual agreement for a length of time in the future. During the contract period, all the products in the contract have to be provided exclusively by these selected suppliers and all the agreed prices will be fixed. Obviously, the two-stage sourcing decision process prevents other potential suppliers, who can offer better deals, from getting the job order. Furthermore, it emphasizes only on price, and treats delivery date as the second prioritized objective. As we all know, in some occasions, especially in the busy periods, customer can be less sensitive to price and more concerned about delivery date.

A new sourcing mechanism is therefore envisioned. Instead of maintaining relatively static and limited relationship, customer and supplier in this sourcing mechanism collaborate in a more flexible and dynamic manner. Facing a particular demand, a customer initiates an auction opened to all suppliers. Suppliers are allowed to submit multiple options, each representing a price-delivery date tuple. These options can be presented as a Price Delivery Date Frontier (PDDF) (see figure 1). An option from a supplier that maximizes customer's utility will be selected.

[Insert figure 1 about here]

In order to generate a PDDF, among other factors, suppliers should be aware of their cost for each delivery date option. Scheduling the new demand in a particular production period (delivery date) could affect cost. For example, if the new demand is scheduled in a busy period, it can make some of the previously committed demand late. Because of that, the penalty will be incurred. Moreover, scheduling the new demand in a period in which there is no same product type being produced could cost additional setup cost. In this paper, we

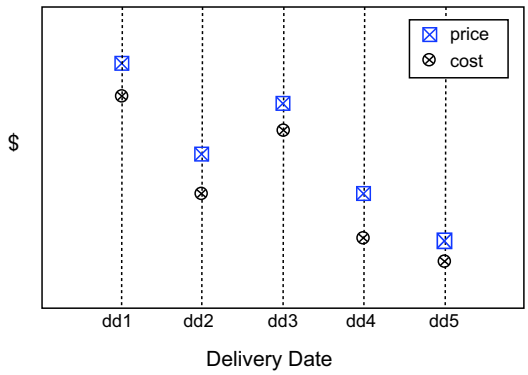

Figure 1: The Price-Delivery Date Bid Frontier and the Cost-Delivery Date Frontier

are interested in a supplier whose manufacturing system can be represented by the capacitated multi-item lot-sizing model with backorders. By solving the lot-sizing problem, the optimal production plan which minimizes the total cost, and hence, the cost-delivery date tuple, can be obtained. Illustrated as the circles in figure 1, the cost-delivery date tuples form the Cost-Delivery Date Frontier (CDDF).

This paper is organized as follows. Section 2 presents the mathematical formulation of the lot-sizing problem under consideration. In section 3, a family of valid inequalities for the problem is introduced. With the aim to add only effective valid inequalities to the problem, section 4 describes three valid inequality selection strategies. Computational results are then reported in section 5. We conclude this paper in section 6.

2 Mathematical formulation

Consider the situation in which the supplier receives an RFQ containing demand for product j' with the quantity of q' units. Suppose the supplier desires to find a cost-delivery date tuple $(t', z^*(t'))$ on the CDDF, where t' and $z^*(t')$ are a potential delivery date (period) and the corresponding minimum cost, respectively. Let $e_{j't'}$ be the existing committed demand quantity for product j' in period t'. The total demand for product j' in period t' is denoted by $d_{j't'}$, and it can be calculated as:

$$d_{j't'} = e_{j't'} + q' \tag{1}$$

For demand of any product $j \neq j'$ in period $t \neq t'$, the total demand d_{jt} can simply be expressed as:

$$d_{jt} = e_{jt} \tag{2}$$

The considered problem is called Capacitated Multi-Item Lot-Sizing Problem with Backorder (CMB). The notations used in this research are summarized in table 1. Following is the MIP formulation representing the CMB.

$$\min \quad \sum_{j\in\mathbb{J}}\sum_{t\in\mathbb{T}}(f_j y_{jt} + h_j s_{jt} + b_j u_{jt}) \tag{3}$$

subject to

$$x_{jt} + s_{j(t-1)} + u_{jt} = d_{jt} + s_{jt} + u_{j(t-1)}, \qquad \forall j \in \mathbb{J}, \forall t \in \mathbb{T} \tag{4}$$

$$x_{jt} \leq (\frac{C_t}{p_j}) y_{jt}, \qquad \forall j \in \mathbb{J}, \forall t \in \mathbb{T} \tag{5}$$

$$\sum_{j\in\mathbb{J}} x_{jt} p_j, \leq C_t \qquad \forall t \in \mathbb{T} \tag{6}$$

$$\sum_{t\in\mathbb{T}} x_{jt} = \sum_{t\in\mathbb{T}} d_{jt}, \qquad \forall j \in \mathbb{J} \tag{7}$$

$$y_{jt} \in \{0,1\}, \qquad \forall j \in \mathbb{J}, \forall t \in \mathbb{T} \tag{8}$$

$$x_{jt}, s_{jt}, u_{jt} \geq 0, \qquad \forall j \in \mathbb{J}, \forall t \in \mathbb{T} \tag{9}$$

The objective function (3) is to minimize the total setup cost, inventory holding cost, and backorder cost. Equations (4) represent inventory balance constraints. The setup state variable y_{jt} is controlled by the logical constraints (5). Inequalities (6) limit the maximum total production of a period. Constraints (7) ensure that, eventually, provided that T is large enough, all demands will be fulfilled. Constraints (8) restrict the setup state variable y_{jt} to be binary number, while constraints (9) enforce production, inventory and backorder levels to be nonnegative. We call the solution set satisfying (4) – (9) as $\mathbb{X}^{CMB}$.

[Insert table 1 about here]

3 Valid inequalities

Capacitated multi-item lot-sizing problem, in general, is a difficult problem. Wolsey (1998) shows that a basic type of it is $\mathcal{NP}$-hard. There has been great effort, however, by researchers to solve many variants of the capacitated multi-item lot-sizing problem via the applications of valid inequalities.

Through the study of the polyhedral structure of an MIP formulation of the single-item model, Barany et al. (1984) derives valid inequalities for the multi-

Sets and indices

Symbol	Description
$\mathbb{Z}_+$	$= \{0, 1, \ldots\}$, set of nonnegative integral numbers
$\mathbb{T}$	$= \{1, 2, \ldots, T\}$, set of production periods
$\mathbb{J}$	$= \{1, 2, \ldots, J\}$, set of products supplier can produce
t	time period in $\mathbb{T}$
j	product type in $\mathbb{J}$

Parameters

Symbol	Description
h_j	per unit per period inventory holding cost of product j
b_j	per unit per period backorder cost of product j
d_{jt}	units of demand for product j in period t
C_t	production capacity limit of period t (in units of time)
p_j	unit production time required to produce product j

Decision variables

Symbol	Description
x_{jt}	units of product j produced in period t
y_{jt}	setup status for product j in period t; $y_{jt} = 0$ if product j is not produced in period t, $y_{jt} = 1$ otherwise
s_{jt}	nonnegative inventory of product j in period t
u_{jt}	nonnegative backordered units of product j in period t

Table 1: Notations used in this paper

item model. The class of valid inequalities, which is called the (l, S) inequalities, 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 backorders. Other research that employ the same approach for other capacitated multi-item models include Constantino (1996), Leung et al. (1989), Magnanti and Vachani (1990), and Pochet and Wolsey (1991).

Another important research that addresses the uncapacitated lot-sizing problem with backorder is conducted by Agra and Constantino (1999). They are able to prove that in the optimal solution, a demand will only be satisfied exclusively by stock, by production in that period, or by backorder. Also, they find that if the machine is setup in a given period, the demand of that period is satisfied solely by production in that period. With these properties, they develop valid inequalities for uncapacitated lot-sizing problems. In the capacitated case, however, limited capacity could force the demand to be satisfied by more than single source. Adding the uncapacitated case's valid inequalities could exclude a portion of the solution set of the capacitated problem. Therefore, Agra and Constantino's valid inequalities for the uncapacitated case are not necessarily valid for the capacitated case. A survey on the research involving the application of valid inequalities to lot-sizing models can be found in Pochet (1995).

Following, a family of valid inequalities for CMB, is presented.

Proposition 1 *The inequalities*

$$x_{jt} \leq d_{jt} y_{jt} + s_{jt} + u_{j(t-1)}, \qquad j \in \mathbb{J}, t \in \mathbb{T} \tag{10}$$

are valid inequalities for $\mathbb{X}^{CMB}$.

Proof Consider two possible cases.

Case 1 ($y_{jt} = 0$) Since $y_{jt} = 0$, it can be concluded that $x_{jt} = 0$. The inequalities can now be rewritten for this situation as $0 \leq 0 + s_{jt} + u_{j(t-1)}$. Obviously, the inequalities are valid because $s_{jt} \geq 0$ and $u_{jt} \geq 0$ for all $j \in \mathbb{J}$ and for all $t \in \mathbb{T}$.

Case 2 ($y_{jt} = 1$) In this situation, the inequalities can be rewritten and $x_{jt} \leq d_{jt} + s_{jt} + u_{j(t-1)}$. Recall that, for any $j \in \mathbb{J}$, $t \in \mathbb{T}$, the inventory balance constraint is $x_{jt} + s_{j(t-1)} + u_{jt} = d_{jt} + s_{jt} + u_{j(t-1)}$. It is clear that $x_{jt} \leq d_{jt} + s_{jt} + u_{j(t-1)}$ must hold, hence the inequalities are again valid. □

Because this family of valid inequalities is derived from the lot-sizing problem with backorders, the valid inequalities shown in (10) will be called the *LSB inequalities*. Furthermore, since a valid inequality can be identified by two indices, j and t, we will denote LSB($\tilde{j}, \tilde{t}$) an LSB inequality with $j = \tilde{j}$ and $t = \tilde{t}$.

4 Valid inequality selection strategies

By adding valid inequalities to the model, we increase the problem size. Hence, unless the valid inequalities are very effective, the computing time required to solve the problem would generally increase as a result. Careful consideration on which valid inequalities to be added to the model needs to be taken. Three valid inequality selection strategies are developed and described in detail in section 4.1–4.3. These selection strategies can be applied to any family of valid inequalities. But in this research, however, we apply them only to the LSB inequalities.

4.1 All Optimal Point Violating Cuts (All-V)

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

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

Step 2 Add to CMB all the LSB inequalities

$$x_{jt} \leq d_{jt} y_{jt} + s_{jt} + u_{j(t-1)}, \qquad j \in \mathbb{J}, t \in \mathbb{T}$$

that

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

Basically, All-V solves the LP relaxation first, then it adds all the violated valid inequalities to the model. An obvious advantage of All-V strategy is that it requires very modest amount of time to select the valid inequalities.

For CMB, the setup variables y_{jt}'s are discrete. Hence, the solution of the relaxation could contain a number of fractional $\bar{y}^*_{jt}$'s. In Sections 4.2 and 4.3, we introduce other two valid inequality selection strategies that try to reduce the number of fractional y_{jt}'s, in hope that with less number of fractional y_{jt}'s, optimal solution could be found faster.

4.2 Most Optimal Point Violation (Most-V)

The Most Optimal Point Violation (Most-V) follows an iterative procedure. Again, let RCMB be the LP relaxation of CMB to which, over the loops, valid inequalities will be added. Also, let $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$ and v^k indicate RCMB's optimal solution and the maximum violation found in the k-th loop, respectively. Following is the description of Most-V strategy.

Step 0 Set $k = 0$

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

Step 2 For all $j \in \mathbb{J}$ and $t \in \mathbb{T}$, find

$$v^k = \max_{j\in\mathbb{J}, t\in\mathbb{T}} \{\bar{x}^{k*}_{jt} - (d_{jt}\bar{y}^{k*}_{jt} + \bar{s}^{k*}_{jt} + \bar{u}^{k*}_{j(t-1)})\}$$

and the corresponding

$$(j', t') = \arg\max_{j\in\mathbb{J}, t\in\mathbb{T}} \{\bar{x}^{k*}_{jt} - (d_{jt}\bar{y}^{k*}_{jt} + \bar{s}^{k*}_{jt} + \bar{u}^{k*}_{j(t-1)})\}$$

Step 3 If $v^k \leq 0$, STOP. Else, go to Step 4.

Step 4 Add LSB(j', t') to RCMB

Step 5 Go back to Step 1

Step 1 to step 5 form a loop, in which, RCMB is solved, then the most violated valid inequality is selected and added to the formulation. In a k-th loop, if the LSB inequalities are used up or there is no violation (i.e. $v^k \leq 0$), the process terminates.

In each loop, through the measure of violation v^k, Most-V selects only the valid inequality with the highest potential to cut off the most number of fractional solution points. After successive additions of valid inequalities and the process terminates, we hope that the final solution to the RCMB should be very similar to the optimal integer solution.

By using Most-V, however, RCMB has to be iteratively solved then the most violated LSB inequality is determined and added to the problem. If Most-V has to add a very large number of inequalities, which is often the case for many valid inequality families, the computing time required to run Most-V could become significant. Another possible drawback of Most-V, and as well as All-V for that matter, is that by cutting off RCMB's optimal point with a valid inequality, even more number of extreme points can be created. Therefore, if RCMB is solved using the Primal Simplex Method, which has to move from one extreme point to another until optimal point is found, the computing time to solve the LP in the children nodes could increase as a result. Acknowledging this potential drawback, the following strategy is developed.

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

Assuming that, at each node in the branch-and-bound tree, we solve an LP using Primal Simplex Algorithm (PSA). Starting from an initial feasible point, PSA moves to an adjacent extreme point until the optimal point is found. Using PSA, the LP optimal path is therefore determined. The idea behind Max-XP is to select the valid inequality that cuts off maximum number of the extreme points, counting backward, along the optimal path, from the optimal point to the initial point, without skipping.

This is done with two purposes. Given the fact that the PSA moves from one extreme point to another with non-deteriorating solution quality, the points closer to the optimal solution are likely to be of higher quality than the ones farther away. 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 some of the lower objective solutions from the feasible region. Left with higher objective solutions in the feasible region, we can achieve smaller optimality gap and hence reduce the number of nodes need to be examined.

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 children nodes.

[Insert table 2 about here]

Notations necessary to describe Max-XP can be found in table 2. Following is the description of the Max-XP valid inequality selection strategy.

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

Step 1 Increment k, set $\gamma^k_{max} = 0$, solve RCMB to 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 on 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 $j \in \mathbb{J}$ and $t \in \mathbb{T}$, *pop* extreme point information object from σ^k one by one, for each i extreme point information object, calculate

$$v^{ki}_{jt} = \bar{x}^{ki}_{jt} - (d_{jt}\bar{y}^{ki}_{jt} + \bar{s}^{ki}_{jt} + \bar{u}^{ki}_{j(t-1)})$$

then add v^{ki}_{jt} to v^k_{jt} by,

$$v' = v^k_{jt}$$

Notation	Description
i	number of extreme points counted
k	number of valid inequalities being added
RCMB	LP relaxation of CMB to which valid inequalities are added
I^k	number of iterations needed to solve RCMB in k-th loop
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_{jt}^{ki}	violation of the LSB(j,t) by $(\bar{x}^{ki}, \bar{y}^{ki}, \bar{s}^{ki}, \bar{u}^{ki})$
v_{jt}^{k}	total sum of v_{jt}^{ki}'s
γ_{jt}^{k}	number of extreme points LSB(j,t) can cut off in 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 keeping information of the valid inequalities that can cut off maximum number of extreme points in k-th loop

Table 2: Notations for Max-XP strategy

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

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

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

Step 7 Go back to Step 1.

At step 1, RCMB 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 on 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. Much of the calculation is done at Step 5. At this step, Max-XP goes through all the LSB(j,t)'s to determine γ_{max}^k and v_{jt}^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 $(\bar{x}^{k*}, \bar{y}^{k*}, \bar{s}^{k*}, \bar{u}^{k*})$.

Max-XP strategy is designed to serve the two purposes, as mentioned earlier. However, Max-XP 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 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 different from Most-V, Max-XP needs to solve RCMB twice for each addition of a valid inequality, the first time for determining the total number of iterations needed, and the second time for the information collection of the last P extreme points. Moreover, by using Max-XP, since the information regarding the last P extreme points have to be kept, a tremendous proportion of time is required in the input/output process. Lastly, in order to find the total sum of violations in any k-th loop, Max-XP has to do more calculations than All-V

and Most-V do. While Most-V does a single v_{jt}^{k} calculation for each LSB(j,t), Max-XP has to do P times of the v_{jt}^{ki} calculation before being able to determine the total sum of the violation v_{jt}^{k}. Based upon our experience with Max-XP, it is observed that most of the time used 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.

5 Implementation and computational results

Branch-and-Bound is the most common algorithm used for solving MIP problems. In order to solve CMB, we also employ Branch-and-Bound algorithm. Plenty of commercial MIP optimizer softwares (or solvers) are available. Most of these solvers provide users with a number of implementation options. For solving MIP, users can specify the types of generic cuts and the nodes in the branch-and-bound tree at which cuts will be added.

5.1 Solving strategies considered

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

In solving an IP or MIP problem, CPLEX, by default, will automatically determine which types of generic cuts to be added and the nodes in the branch-and-bound tree at which the cuts will be added. While the default setting performs acceptably in general cases, users are provided with solving options, in case 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 Magee and Glover (1996) and Wolsey (1998). 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, there are five implementation options. Three of them are the All-V, Most-V, and Max-XP strategies presented in section 4.1– 4.3. The last two options are called No-LSB and All-LSB, by which *none* and *all* of the LSB inequalities are added, respectively. If they are added, the LSB inequalities will be added only at the root node. Since there are three options in the addition of the CPLEX cuts and five options in the addition of the LSB inequalities, in total, there are fifteen solution strategies in solving CMB (see figure 2).

[Insert figure 2 about here]

We denote α/β the solution strategy that employs α CPLEX cut option and β LSB inequalities selection strategy. 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

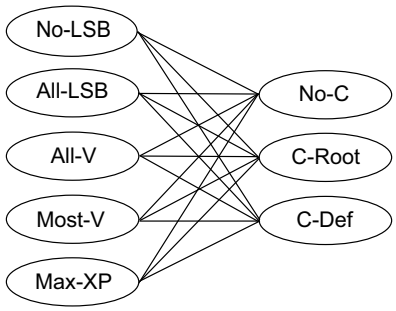

Figure 2: Fifteen solving strategies considered

to implement the All-V, Most-V, and Max-XP. The selected LSB inequalities are added to the GAMS model through the Perl script. When the solver is in need, 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 the Pentium III 800 MHz PC machine with Linux operating system.

5.2 Test problems

We consider a manufacturing system producing five product types ($J = 5$). Each unit of a product type j 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

$$D(t) = \min\{0, D_1 - \delta t\} \tag{11}$$

where D_1 and $D(t)$ are the total demand units of all the five products in period 1 and period $t > 1$, respectively. The decreasing rate of demand is positive and it is represented by δ. Please note that $D(t)$ can also be expressed as $D(t) = \sum_{j=1}^{J} d_{j,t}$. To assign a value to a $d_{j,t}$, we pick a random number r_j in $(0, 1)$ for each job j, then assign

$$d_{j,t} = (\frac{r_j}{\sum_{j=1}^{J} r_j}) \cdot D(t) \tag{12}$$

To be sure that eventually the demands will be fulfilled, $T = 40$ is selected. 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 j, the late penalty b_j, the holding cost h_j, and a setup cost f_j are picked from uniformly distributed ranges $[\bar{b}_l, \bar{h}_h]$, $[\bar{h}_l, \bar{b}_h]$, and $[\bar{f}_l, \bar{f}_h]$, respec-

tively. In our experiment, we assign values to all of the mentioned parameters and they are summarized in table 3.

[Insert table 3 about here]

5.3 Computational results

Ten test problems are generated. These problems are used to evaluate the fifteen solving strategies described in section 5.1. A solving process stops if one percent optimality gap is achieved, or if the maximum computing time limit of 1800 seconds is reached.

We solve each of the ten test problems using the fifteen solution strategies described in section 5.1. The computing time used by each strategy is recorded and shown in table 4. For a given problem, we compare the computing times used by the fifteen 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. The normalized computing times are shown in table 5.

It can be observed from table 4 that none of the ten test problems can be solved to one percent optimality gap within half an hour time using No-C/No-LSB strategy. On the other hand, by using other solution strategies, the one percent optimality gap can be achieved in eight of the ten problems. The asterisks (*) displayed in table 4 represent the case where a solution strategy cannot reach the one percent optimality gap within half an hour.

In table 5 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 the table, it is found that, except for problem 8, the other nine test problems can be solved fastest by the solving strategies that add the LSB inequalities through All-V. Comparing two solving strategies No-C/All-V and C-Root/No-LSB, where either CPLEX cuts or LSB inequalities are added, the results show that except for Problem 10, the other nine problems can be solve faster by No-C/All-V. Another interesting observation is that Max-

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 3: Parameter values used in the experiment

XP does not perform particularly well. This is mainly because it requires a great deal of time for the input/output process while it tries to collect extreme point information. Focusing on the last column of table 5, it is clear that, on average, the three fastest solving strategies (i.e. No-C/All-V, C-Root/All-V, and C-Def/All-V) add LSB inequalities only through All-V strategy. Furthermore, on average, No-C/All-V is the fastest solving strategy.

[Insert table 4 about here]

[Insert table 5 about here]

6 Conclusion

We consider the sourcing process in which suppliers (bidders) are allowed to submit Price-Delivery Date Frontier (PDDF) as a bid. A PDDF contains multiple price-delivery date options. Customer chooses one of the options submitted by all the suppliers that maximizes its utility. As cost is usually one of the most important factor in determining the price, this paper addresses the problem of how to quickly and accurately determine a cost-delivery date tuple in order to construct the Cost-Delivery Date Frontier (CDDF). We focus our attention on the capacitated multi-item lot-sizing systems with backorders. We introduce a family of valid inequalities, called the LSB inequalities. Also, three valid inequality selection strategies are proposed. Through the selection strategies, the valid inequalities can be determined and added to the problem's formulation it is solved. A commercial solver, called CPLEX, is used in our experiments, by which some generic cuts can be generated and added to the nodes in the branch and bound tree. The computational results strongly suggest that, to solve the problem (i.e. to determine a cost-delivery date tuple in the CDDF) quickly, LSB inequalities should be applied through a proposed selection strategy called All-V, without applying any of the CPLEX generated cuts.

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-LSB	479	148	1800*	1800*	46	4	146	22	652	1235
	All-V	408	77	1800*	1800*	26	5	88	17	497	780
	Most-V	950	120	1800*	1800*	54	41	152	51	554	1152
	Max-XP	885	233	1800*	1800*	180	150	317	198	681	1351
C-Root	No-LSB	552	100	1800*	1800*	118	12	106	31	1265	595
	All-LSB	326	111	1800*	1800*	190	5	145	25	645	1507
	All-V	176	89	1800*	1800*	73	16	154	15	503	290
	Most-V	910	17	1800*	1800*	101	42	155	48	643	747
	Max-XP	480	548	1800*	1800*	228	150	321	194	711	874
C-Def	No-LSB	548	75	1800*	1800*	119	18	97	29	1800*	593
	All-LSB	322	98	1800*	1800*	188	5	129	24	1105	1800*
	All-V	176	53	1800*	1800*	72	19	138	14	1800*	291
	Most-V	933	64	1800*	1800*	102	42	144	47	1800*	728
	Max-XP	480	312	1800*	1800*	228	150	311	193	1138	873

Table 4: Computing times (in seconds) required to solve the problems

Problem		1	2	3	4	5	6	7	8	9	10	Average
No-C	No-LSB	10.21	34.23	n/a	n/a	69.52	455.70	20.49	127.12	3.63	6.22	72.71
	All-LSB	2.72	2.81	n/a	n/a	1.78	1.00	1.66	1.53	1.31	4.26	1.71
	All-V	2.31	1.47	n/a	n/a	1.00	1.19	1.00	1.23	1.00	2.69	1.19
	Most-V	5.39	2.28	n/a	n/a	2.07	10.47	1.73	3.62	1.12	3.98	3.07
	Max-XP	5.02	4.44	n/a	n/a	6.95	38.01	3.61	13.96	1.37	4.67	7.80
C-Root	No-LSB	3.13	1.89	n/a	n/a	4.57	3.13	1.21	2.16	2.55	2.06	2.07
	All-LSB	1.85	2.11	n/a	n/a	7.34	1.18	1.65	1.77	1.30	5.20	2.24
	All-V	1.00	1.70	n/a	n/a	2.82	3.95	1.75	1.06	1.01	1.00	1.43
	Most-V	5.16	2.23	n/a	n/a	3.91	10.54	1.77	3.38	1.29	2.58	3.09
	Max-XP	2.72	10.42	n/a	n/a	8.79	37.96	3.65	13.73	1.43	3.02	8.17
C-Def	No-LSB	3.11	1.42	n/a	n/a	4.62	4.63	1.10	2.08	3.63	2.05	2.26
	All-LSB	1.82	1.87	n/a	n/a	7.26	1.15	1.46	1.67	2.23	6.22	2.37
	All-V	1.00	1.00	n/a	n/a	2.79	4.85	1.57	1.00	3.63	1.00	1.69
	Most-V	5.29	1.21	n/a	n/a	3.93	10.53	1.64	3.34	3.69	2.52	3.22
	Max-XP	2.72	5.93	n/a	n/a	8.81	37.96	3.54	13.65	2.29	3.02	7.79

Table 5: Normalized computing times

Acknowledgements

This material is based upon work supported by a National Science Foundation Grant (DMI 9800449) to the second author.

References

AGRA, A. and CONSTANTINO, M., 1999, Lotsizing with backlogging and start-ups: the case of Wagner-Whitin costs, *Operations Research Letters*, 25, 81–88

BARANY, I., ROY, T. J. V., and WOLSEY, L. A., 1984, Strong formulations for multi-item capacitated lot sizing, *Management Science*, 30(10), 1255–1261

CONSTANTINO, M., 1996, A cutting plane approach to capacitated lot-sizing with start-up costs, *Mathematical Programming*, 75, 353–376

LEUNG, J., MAGNANTI, T., and VACHANI, R., 1989, Facets and algorithms for capacitated lot-sizing, *Mathematical Programming*, 45, 331–359

MAGEE, T. M. and GLOVER, F., 1996, Integer programming. In M. Avriel and B. Golany (eds.), *Mathematical Programming for Industrial Engineers*, Vol. 20 of *Industrial Engineering* (Marcel Dekker, Inc.), Chapt. 3, pp. 123–269

MAGNANTI, T. and VACHANI, R., 1990, A strong cutting plane algorithm for production scheduling with changeover costs, *Operations Research*, 38, 456–473

POCHET, Y. and WOLSEY, L., 1991, Solving multi-item lot-sizing problems using strong cutting planes, *Management Science*, 37, 53–67

POCHET, Y. and WOLSEY, L. A., 1995, Algorithms and reformulations for lot sizing problems. In W. Cook, L. Lovasz, and P. Seymour (eds.), *Combinatorial optimization: papers from the DIMACS special year*, Vol. 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (American Mathematical Society), Chapt. 6, pp. 245–293

WOLSEY, L. A., 1998, *Integer Programming* (Wiley-Interscience)