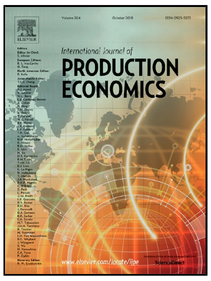


Inventory Control with Flexible Demands: Cyclic Case with Multiple Batch Supply and Demand Processes

Matthew E.H. Petering, Xi Chen, Wen-Huan Hsieh

PII:	S0925-5273(18)30405-5
DOI:	10.1016/j.ijpe.2018.09.035
Reference:	PROECO 7189
To appear in:	*International Journal of Production Economics*
Received Date:	02 March 2017
Accepted Date:	30 September 2018

Inventory Control with Flexible Demands: Cyclic Case with Multiple Batch Supply and Demand Processes

Matthew E. H. Petering[a*], Xi Chen[b], Wen-Huan Hsieh[a]

[a] University of Wisconsin—Milwaukee, Department of Industrial and Manufacturing Engineering, P.O. Box 784, Milwaukee, Wisconsin 53201, USA. *Email addresses*: mattpete@uwm.edu, j4984522@gmail.com.

[b] University of Michigan—Dearborn, Department of Industrial and Manufacturing Systems Engineering. *Email address*: xichenxi@umich.edu.

[*] Corresponding author. Tel.: +1-414-229-3448; fax: +1-414-229-6958.

Inventory Control with Flexible Demand: Cyclic Case with Multiple Batch Supply and Demand Processes

Abstract

We introduce, and present methods for solving, the *cyclic inventory control problem with multiple flexible batch supply and demand processes*. The objective of this new problem is to minimize the average or maximum amount of inventory of a single item that is held during a cycle of given length in a buffer whose stock is replenished by multiple batch supply processes and consumed by multiple batch demand processes. The problem is noteworthy in that the decision maker has control over the timing and lot sizes of all supply and demand processes subject to (1) minimum frequency and batch size requirements for each demand process and (2) maximum frequency and batch size capabilities for each supply process. Thus, demand is *flexible*; it is a control action that the decision maker applies to optimize the system. We model this deterministic problem as an integer linear program; obtain theoretical insights concerning problem feasibility and solution optimality; and develop three heuristic methods for attacking large problem instances. Extensive experiments compare five methods for attacking the problem: pure integer programming using IBM ILOG CPLEX; integer programming where CPLEX is given an initial feasible solution; a genetic algorithm; simulated annealing; and a random algorithm. We find that CPLEX is a good option for solving small problem instances and the proposed genetic algorithm outperforms CPLEX on larger instances.

Keywords

Inventory control, flexible demand, cyclic scheduling, just-in-time, batch processing, joint economic lot sizing

1. Introduction

Inventory is caused by supply and demand not being synchronized and is a fundamental aspect of manufacturing and distribution systems. The lack of synchronization is often of two types: the timing of the supply and demand processes differs, or the amount supplied and demanded at one time (i.e. the supply and demand *batch size*) differs. These mismatches are due to processing/transit times, production occurring in batches, and/or demand uncertainty (Graves, 1987). In this paper, we introduce a new inventory problem faced by a facility that acts as a buffer between multiple supply and demand processes that are asynchronous due to processing/transit times and batch production, but not demand uncertainty.

Our goal is to find a cyclic schedule that minimizes work-in-process (WIP) inventory. This concern is consistent with the Just-In-Time (JIT) principles adopted by many companies. JIT's practice of inventory reduction has brought many benefits such as reducing lead times, enabling faster feedback on quality, improving supplier relationships, and empowering employees (Singh 1990, Malakooti 2013). However, the restrictions imposed by batch production and production capacity limits add to the challenge of minimizing inventory. Parts are often manufactured in fixed batches due to high machine setup costs or the need to fill up containers or truckloads. Moreover, production quantities may be limited by machine and/or shift capacity (Li et al., 2004). As a result, minimizing WIP inventory requires coordinating the supply and demand processes while accounting for their capacity constraints and production requirements.

The coordination between supply and demand processes is further complicated if multiple suppliers or demanders are present and if they are heterogeneous. In this paper, we consider a problem where parallel heterogeneous suppliers supply a common part to multiple independent, heterogeneous demanders. This problem is relevant to manufacturing systems where the supply and demand processes represent stages of manufacturing with parallel machines, and to supply chain systems where the supply and demand processes link a central storage facility to supplier and buyer facilities. We provide more details of these applications below.

1.1. Problem description

We now formally introduce the problem under investigation. This problem is called the *cyclic inventory control problem with multiple flexible batch supply and demand processes* (CICP-MFBSDP). Consider an infinite-capacity *buffer* or *facility* that serves as a temporary storage area for a single type of discrete *product* (i.e. item, SKU, part). The stock in the buffer is consumed by *D demanders* and replenished by *S suppliers*. Time is discretized into *periods*. The demand associated with each demander d is defined by two parameters—a *maximum inter-demand time* DT_d and a (*minimum*) *demand quantity* DQ_d. Demander d needs to be given a batch of at least DQ_d units from the buffer every DT_d periods or more often for all d. The supply associated with each supplier s is defined by two parameters—a *minimum inter-supply time* ST_s and a (*maximum*) *supply quantity* SQ_s. Supplier s is capable of delivering a batch of at most SQ_s units to the buffer every ST_s periods or less often for all s. Assume that supplies are received at the beginning of a period and are followed immediately by demands. The amount left over after the demand is satisfied is held as inventory for that period. The operations are cyclic, repeating every T periods where $T \geq \max_d\{DT_d\}$ and $T \geq \max_s\{ST_s\}$. In other words, every supply and demand process repeats after a T-period cycle. In addition, period 1 immediately follows period T. Thus, the inventory level during period 1 equals the inventory level during period T, plus the total amount supplied to the buffer at the beginning of period 1, minus the total amount taken from the buffer at the beginning of period 1. The demand (supply) timing and batch sizes for each demander (supplier) are decided by the system manager subject to the above requirements (capabilities). The goal is to feasibly satisfy the demands with the available supplies—i.e. to keep the buffer inventory level at least 0 during every period in the cycle—while minimizing the buffer's average or maximum inventory level during the cycle.

Figure 1 depicts this inventory system. The solid rectangle in the middle represents the buffer. The S suppliers—each with a unique minimum inter-supply time ST_s and maximum supply quantity SQ_s—are shown in dashed rectangles on the left. The D demanders—each with a unique maximum inter-demand time DT_d and minimum demand quantity DQ_d—are shown in dashed rectangles on the right.

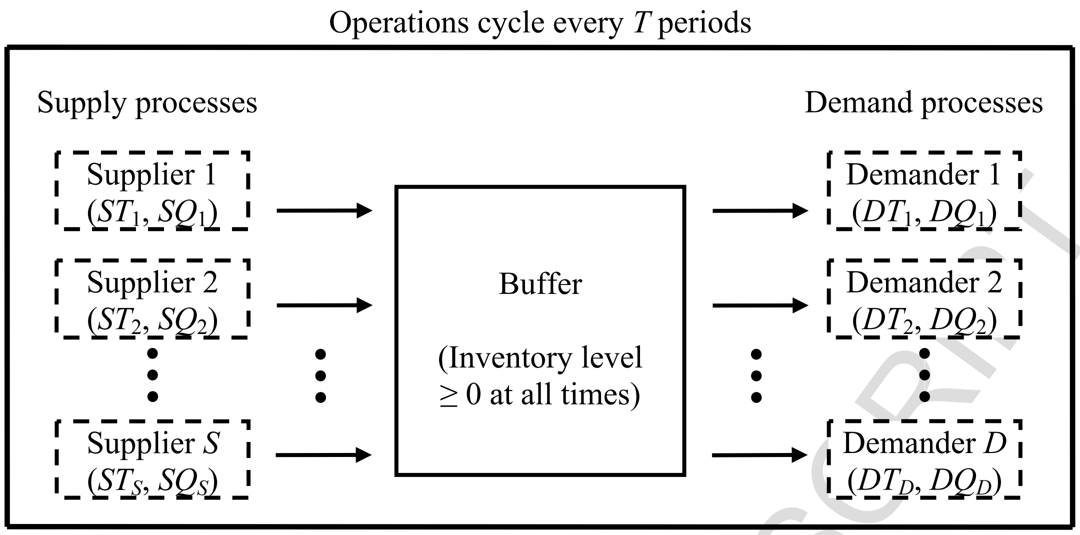

Operations cycle every T periods
Supply processes
Demand processes
Supplier 1 (ST_1, SQ_1)
Supplier 2 (ST_2, SQ_2)
Supplier S (ST_S, SQ_S)
Buffer
(Inventory level ≥ 0 at all times)
Demander 1 (DT_1, DQ_1)
Demander 2 (DT_2, DQ_2)
Demander D (DT_D, DQ_D)

Figure 1. System under investigation

The CICP-MFBSDP is defined so as to represent a variety of microscopic or macroscopic industrial activities. On a microscopic level, the problem could represent a buffer that sits in the middle of a two-stage manufacturing process in a factory. In this case, there are parallel machines at the first stage with heterogeneous capabilities (processing times and batch sizes) that produce the same part that feeds the machines at the second stage. Each second-stage machine produces a different product using the part supplied and therefore has a different cycle time DT_d and/or minimum batch size DQ_d. On a macroscopic level, the problem could represent a two-stage supply chain with parallel suppliers supplying a single product to a retailer that has multiple retail stores, each facing a different demand. The suppliers first deliver their products to a distribution center (i.e. the buffer); the products are then distributed to the various retail stores. A single specialized truck, with limited capacity SQ_s and minimum round-trip travel time ST_s, transports units from supplier s to the distribution center for each s.

The demand processes in the CICP-MFBSDP are somewhat unusual in that batches of DQ_d-1 or fewer units—even multiple such batches provided in different periods—are entirely useless to demander d. On the other hand, batches of size DQ_d or more units are equally useful; demander d receives no immediate or future benefit, and no penalty, by receiving strictly more than DQ_d units at a time. In other words, any batch of DQ_d-1 or fewer units sent to demander d is immediately discarded or returned to the buffer, and any batch of DQ_d or more units is immediately consumed in its entirety by demander d. Also, demander d only needs a batch of the product at least once every DT_d periods; there is no benefit or penalty for demander d if he/she receives it more often than that. These assumptions apply to systems with the following features. First, each demander is unable to store the product due to (i) severely limited storage space; (ii) the nature of the product as a perishable, easily contaminated, or highly toxic material; or (iii) the adoption of JIT principles which require inventory minimization. Second, each demander d runs a batch process of long duration that requires exactly DQ_d units. Examples of such a process might be (a) a lengthy assembly operation that needs DQ_d identical units (e.g. gears, axles, etc.) to be present in order for assembly to commence; (b) the testing and/or calibration of a product that contains DQ_d units; (c) an expensive batch inspection process that is designed for exactly DQ_d units; (d) a batch chemical process in which exactly DQ_d units should be present to optimize the ensuing chemical reaction; (e) the installation of a set of DQ_d batteries that have a useful lifetime of DT_d periods; or (f) the continued survival of a medical patient who requires a blood transfusion involving DQ_d bags of blood every DT_d periods or more often.

The requirement that operations repeat every T periods is imposed by managers who are interested in system reliability and predictability. Without this assumption, the system state could evolve in a complex manner from time 0 to infinity. A value of T that agrees with the length of one or more work shifts might be appropriate; it could be a multiple of seven if time is measured in days or a multiple of 24 if time is measured in hours.

1.2. Illustrative example

Tables 1 and 2 illustrate the problem at hand for a case in which $D = S = 3$. The input parameters for this instance are shown in Table 1. Table 2 shows a feasible solution for this problem instance. In this solution, demander 1 is given 2 units at the beginning of each of the periods T3, T5, T7, and T10; demander 2 is given 4 units at the beginning of periods T2, T4, T6, T8 and T10; and demander 3 is given

2 units at the beginning of periods T4 and T10. Note that the batch sizes given to the demanders—2, 4, and 2 units—are greater than or equal to the values of DQ_1, DQ_2, and DQ_3 respectively. Also, the time that elapses between consecutive demand occurrences never exceeds the values of DT_1, DT_2, and DT_3—3, 2, and 6 periods—for demanders 1, 2, and 3 respectively. In the solution, supplier 1 replenishes the buffer with 4, 4, 4, 1, and 3 units at the beginning of periods T1, T3, T5, T7, and T9; supplier 2 replenishes the buffer with 3 and 3 units at the beginning of periods T2 and T7; and supplier 3 replenishes the buffer with 4, 3, and 3 units at the beginning of periods T1, T4, and T8. Note that the amount delivered by the suppliers never exceeds the values of SQ_1, SQ_2, and SQ_3—4, 3, and 5—for suppliers 1, 2, and 3 respectively. Also, the time that elapses between consecutive supply occurrences is never less than the values of ST_1, ST_2, ST_3—2, 5, and 3 periods—for suppliers 1, 2, and 3 respectively. A zero in Table 2 means that no demand is made or nothing is supplied at the beginning of a period.

Table 1. Input parameters for Illustrative Instance #1

Number of demanders: 3		Number of suppliers: 3	
DT_1: 3	DQ_1: 2	ST_1: 2	SQ_1: 4
DT_2: 2	DQ_2: 4	ST_2: 5	SQ_2: 3
DT_3: 6	DQ_3: 2	ST_3: 3	SQ_3: 5
T = 10			

Table 2. Feasible solution for Illustrative Instance #1

	Period									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Demander 1	0	0	2	0	2	0	2	0	0	2
Demander 2	0	4	0	4	0	4	0	4	0	4
Demander 3	0	0	0	2	0	0	0	0	0	2
Sum up (DI)	0	4	2	6	2	4	2	4	0	8
Supplier 1	4	0	4	0	4	0	1	0	3	0
Supplier 2	0	3	0	0	0	0	3	0	0	0
Supplier 3	4	0	0	3	0	0	0	3	0	0
Sum up (SI)	8	3	4	3	4	0	4	3	3	0
SI-DI	8	-1	2	-3	2	-4	2	-1	3	-8
Inventory held	8	7	9	6	8	4	6	5	8	0
Objective 1 (Total inventory)	61	Objective 2 (Maximum inventory)	9							

The buffer inventory level during each period is shown in the long row at the bottom of Table 2. The sum of these values—61—is the total inventory held during the cycle (i.e. objective 1). The maximum inventory held at any time—the value of objective 2—is 9 units. The goal is to minimize objective 1 or objective 2. Note that the operations cycle every *T* = 10 periods; period T1 immediately follows period T10. In particular, the inventory level during period T1 (= 8) equals the inventory level during period T10 (= 0), plus the total amount supplied to the buffer at the beginning of period T1 (= 8), minus the total amount taken from the buffer at the beginning of period T1 (= 0). The displayed solution is not optimal and is only one of thousands of feasible solutions to this problem instance. The optimal solution—not shown here—has an objective value of 0.

This study is organized as follows. Section 2 reviews the relevant literature. Section 3 presents a mathematical formulation of the problem. Section 4 presents some theoretical insights concerning problem feasibility and solution optimality. Based on these insights we strengthen and modify the initial mathematical formulation. In Section 5, we describe three heuristic methods—a random algorithm, simulated annealing algorithm, and genetic algorithm—that were developed for attacking large problem instances. Section 6 describes the experimental setup, presents the results, and discusses their significance. We conclude in Section 7.

2. Literature review

Our problem is related to the literature on the capacitated lot-sizing problem (CLSP), which seeks to determine the levels and timing of production along a certain planning horizon. The CLSP literature is rich and spans over decades; see Karimi et al. (2003) and Quadt and Kuhn (2008) for reviews. Lot-sizing problems can be categorized into big bucket models and small bucket models. Small bucket models use micro-periods, during which at most one unit of product can be produced. Small bucket models are used to tackle production scheduling problems; see Drexl and Kimms (1997) for a review. Big bucket models allow the production of multiple units in one period and do not incorporate job scheduling decisions.

There is rich literature exploring different CLSP formulations, examining solution bounds, and/or developing solution algorithms; see Kleindorfer and Newson (1975), Eppen and Martin (1987), Trigeiro et al. (1989), Katok et al. (1998), and Sahling et al. (2009) for examples.

Our model considers a single-item two-stage system with parallel machines and heterogeneous workstations. The single-item CLSP has been shown to be NP-hard in general (Florian et al. 1980, Bitran and Yanasse 1982). Within the CLSP literature, there is limited research on production planning problems (i.e., big bucket lot-sizing models) with parallel machines; see Quadt and Kuhn (2008) for a comprehensive review. Hindi (1995) considers a single stage CLSP with multiple items, parallel machines, and no machine setups. Özdamar and Birbil (1998) examine a similar problem with setup times and overtime capacity, which is extended to multiple stages by Özdamar and Barbarosoglu (1999). Belvaux and Wolsey (2000) develop a prototype modeling and optimization system using XPRESS-MP for discrete-time lot-sizing problems including both big bucket and small bucket models. Lee et al. (2009) use a computational grid to solve a material requirements planning (MRP) problem with finite capacity.

The objective of our problem differs from the majority of the CLSP literature which tends to minimize the summation of set-up, inventory, production, and shortage costs. Instead, we seek to minimize the inventory level, which is motivated by Just-In-Time (JIT) principles. JIT is a revolutionary philosophy, originating with the Toyota Company (Ohno, 1988), which aims at eliminating waste in production with the goal of achieving "zero inventory" at its core. There are a number of papers addressing the optimal batch sizes in a JIT environment. For example, using an economic manufacturing quantity framework, Jamal and Sarker (1993) model the optimal production batch sizing problem in a JIT delivery system where raw material arrives continuously at a constant rate. Khan and Sarker (2002) consider a similar problem except that raw materials from suppliers arrive in lots. Both of these two papers model only a single stage of production with a single machine. Sarker and Balan (1999) consider the optimal design of a multi-stage kanban system with linear demand. However, they do not consider the case with parallel machines. McDonald and Karimi (1997) consider the production planning and scheduling of a system with parallel machines of a single-stage, semi-continuous process. Dobson and

Yano (1994) study a batch flow line scheduling problem with multiple products using a pure rotation policy with the goal of minimizing inventory holding cost. They assume demand for each product is continuous and given, and they consider only one machine at each stage.

Our work also differs from the above literature in that the demand from the second production stage arrives in batches. The production planning literature that treats this feature is quite limited. Several papers study the lot sizing problem with batch ordering; see Elmaghraby and Bawle (1972), Webster (1989), and Gaafar (2006) for examples. The majority of this literature considers the lot sizing decision of a single stage. Our paper extends this literature by allowing for the coordination between two stages and parallel stations.

There are a number of papers studying short-term scheduling problems in batch plants in which production occurs only in integer multiples of fixed batch sizes; see Liu and Karimi (2007), Méndez et al. (2001), and Hui and Gupta (2000) for examples. However, this literature mainly adopts the small-bucket formulation with time-related objectives such as minimizing average lateness or makespan. For example, Liu and Karimi (2007) study the performance of several mixed-integer linear programming formulations for a multistage multiproduct production system with non-identical parallel machines and unlimited intermediate storage in order to minimize makespan.

The system we study can also apply to a two-tier supply chain with coordinated inventory replenishment decisions. This is the case especially if the supplier and buyer have worked closely together and have established long-term relationships. This for example is common in the automotive industry (Glock, 2012). Vendor-managed inventory (VMI), for example, is a common practice among such implementations. In our system, the supplier operates parallel workstations supplying to multiple heterogeneous buyers.

The literature on joint economic lot size (JELS) models treats coordinated inventory management between supplier(s) and buyer(s); see Ben-Daya et al. (2008) and Glock (2012) for comprehensive reviews. Most relevant to this study is the JELS literature with parallel supplier workstations and/or multiple buyers. Sawik (2009a) studies a two-tier supply chain with a single buyer and a single supplier who operates multiple parallel production lines. This work is extended by Sawik (2009b) to consider multiple suppliers. Banerjee and Burton (1994) study the JELS problem of a single supplier facing heterogeneous buyers. Chen and Sarker (2010) examine a JELS problem of a single buyer sourcing a product from multiple vendors. Akrami et al. (2006) consider the economic lot sizing and scheduling problem for flexible flow shops with limited intermediate buffers. They consider parallel workshops while assuming equal batch sizes for the same component at different stages.

Our paper extends the JELS literature by integrating the features of parallel workstations at the supplier *and* multiple buyers. Our consideration of batch size constraints and batch ordering processes is also novel to this literature. In addition, our objective is to minimize inventory in a JIT supply chain which differs from the predominant objectives within the JELS literature of minimizing total system cost. Finally, in our problem the demand is *flexible*; it is a control action that the decision maker applies to optimize the system. In most other studies demand is an exogenous (deterministic or stochastic) input to the model. Overall, the problem at hand—which was first introduced in the master's thesis by Hsieh (2015), a co-author of this paper—appears to be new to the production literature.

3. Mathematical formulation of CICP-MFBSDP

Our problem can be modeled as an integer linear program (ILP). The indices, input parameters, and decision variables in this ILP are shown in Tables 3, 4, and 5 respectively. Table 4 shows the parameters that provide input to the math model: the cycle length T; total number of demanders D; maximum inter-demand time for demander d (DT_d); minimum batch size (i.e. demand quantity) needed by demander d (DQ_d); total number of suppliers S; minimum inter-supply time for supplier s (ST_s); maximum batch size (i.e. supply quantity) delivered by supplier s (SQ_s); and weight for objective function component k (W_k). Table 5 displays the decision variables in the model. $DYN_{d,t}$ indicates if demander d is given a batch of sufficient size from the buffer at the beginning of period t or not. $DAmt_{d,t}$ equals the number of units transferred from the buffer to demander d at the beginning of period t. $SYN_{s,t}$ indicates if supplier *s* delivers a batch to the buffer at the beginning of period *t* or not. $SAmt_{s,t}$ equals the number of units delivered to the buffer by supplier *s* at the beginning of period *t*. I_t is the inventory held during period *t*. *IMax* is the maximum inventory level achieved during the *T*-period cycle.

Math Model #1 has two main objectives that can be pursued by setting (W_1, W_2) = (1, 0) or (0, 1). The first (second) objective is to minimize the average (maximum) inventory level during the cycle. A hybrid objective with $W_1 > 0$ and $W_2 > 0$ is also possible. As discussed in Section 1, minimizing inventory is consistent with JIT practices (Dobson and Yano 1994). We note that focusing on inventory minimization may lead to undesirable performance in other aspects, such as excessive production as shown in Observation 1 later in this section. Section 4.4 provides a modification of the problem formulation to mitigate this effect. Constraint (2) ensures that at least one batch of sufficient size is given

Table 3. Indices in Math Model #1

d	demander	(*d* = 1 to *D*)
s	supplier	(*s* = 1 to *S*)
t, u	time period	(*t, u* = 1 to *T*)
k	objective function component	(*k* = 1, 2)

Table 4. Input parameters in Math Model #1

T	Cycle length for the inventory system (i.e. number of periods in the cycle) (integer, > 0).
D	Number of demanders (integer, > 0).
DT_d	Maximum inter-demand time for demander *d* (integer, > 0, ≤ *T*). Maximum number of periods between consecutive batches of sufficient size provided to demander *d*.
DQ_d	Minimum batch size (i.e. demand quantity) needed by demander *d* (integer, > 0).
S	Number of suppliers (integer, > 0).
ST_s	Minimum inter-supply time for supplier *s* (integer, > 0, ≤ *T*). Minimum number of periods between consecutive batches supplied by supplier *s*.
SQ_s	Maximum batch size (i.e. supply quantity) provided by supplier *s* (integer, > 0).
W_k	Weight for objective function component *k* (real, ≥ 0).

Table 5. Decision variables in Math Model #1

$DYN_{d,t}$	= 1 if demander *d* is given a batch of sufficient size at the beginning of period *t* (binary).
$DAmt_{d,t}$	Amount given to demander *d* at the beginning of period *t* (integer, ≥ 0).
$SYN_{s,t}$	= 1 if supplier *s* supplies a batch at the beginning of period *t* (binary).
$SAmt_{s,t}$	Amount supplied by supplier *s* at the beginning of period *t* (integer, ≥ 0).
I_t	Inventory level during period *t* (integer, ≥ 0).
IMax	Maximum inventory level achieved during the cycle (integer, ≥ 0).

to demander d during any DT_d-period time span for all d. Constraint (3) ensures that the batches given to demander d are of sufficient size; the amount given to demander d is at least DQ_d (0) when $DYN_{d,t} = 1$ (0) for all d. This constraint ensures that $DYN_{d,t} = 0$ when $DAmt_{d,t} < DQ_d$. That is, batches of DQ_d-1 or fewer units—even multiple such batches provided in different periods—are entirely useless to demander d. Constraint (4) ensures that at most one batch is supplied by supplier s during any ST_s-period time span. Constraint (5) ensures that the amount supplied by supplier s does not exceed SQ_s in any period and that supplier s does not supply anything at the beginning of period t if $SYN_{s,t} = 0$. Constraints (6-7) ensure that the inventory on hand during each period is properly computed. Constraint (8) ensures that the inventory level in any period does not exceed the maximum inventory level. Constraint (9) requires that no inventory be on hand during the final period. This constraint eliminates symmetries and redundant solutions. The decision variable domains are shown in Table 5. The non-negativity of variable I_t ensures that the inventory level never falls below zero.

Math Model #1:

Minimize: $$(W_1)\sum_{t=1}^{T} I_t + (W_2)(IMax) \qquad (1)$$

Subject to: $$\sum_{u=t}^{t+DT_d-1} DYN_{d,((u-1) \bmod T)+1} \geq 1 \qquad \forall d\,\forall t \qquad (2)$$

$$DAmt_{d,t} \geq (DQ_d)(DYN_{d,t}) \qquad \forall d\,\forall t \qquad (3)$$

$$\sum_{u=t}^{t+ST_s-1} SYN_{s,((u-1) \bmod T)+1} \leq 1 \qquad \forall s\,\forall t \qquad (4)$$

$$SAmt_{s,t} \leq (SQ_s)(SYN_{s,t}) \qquad \forall s\,\forall t \qquad (5)$$

$$I_1 = I_T + \sum_{s=1}^{S} SAmt_{s,1} - \sum_{d=1}^{D} DAmt_{d,1} \qquad (6)$$

$$I_t = I_{t-1} + \sum_{s=1}^{S} SAmt_{s,t} - \sum_{d=1}^{D} DAmt_{d,t} \qquad \forall t : 2 \leq t \leq T \qquad (7)$$

$$I_t \leq IMax \qquad \forall t \tag{8}$$

$$I_T = 0 \tag{9}$$

4. Theoretical insights and further assumptions

We now make several observations concerning problem feasibility and solution optimality, and we use these insights to strengthen and modify Math Model #1. To facilitate the discussion, we introduce four new parameters. Let *$MinNumD_d$* be the minimum number of batches of sufficient size that must be given to demander *d* during a cycle. Let *MinTotalD* be the minimum total number of units demanded during a cycle. Let *$MaxNumS_s$* be the maximum number of batches that can be supplied by supplier *s* during a cycle. Let *MaxTotalS* be the maximum total number of units that can be supplied to the buffer during a cycle. These values can be computed based on the parameters in Table 4 as follows. The brackets in equations (10) and (12) indicate the greatest integer and least integer functions respectively.

$$MinNumD_d = \left\lceil \frac{T}{DT_d} \right\rceil \qquad \forall d \tag{10}$$

$$MinTotalD = \sum_{d=1}^{D} (MinNumD_d)(DQ_d) \tag{11}$$

$$MaxNumS_s = \left\lfloor \frac{T}{ST_s} \right\rfloor \qquad \forall s \tag{12}$$

$$MaxTotalS = \sum_{s=1}^{S} (MaxNumS_s)(SQ_s) \tag{13}$$

We note that, in practice, *T* may be an integer multiple of both DT_d and ST_s for ease of management, which implies that $MinNumD_d = T/DT_d$ and $MaxNumS_s = T/ST_s$. This is a special case of the above setup. In what follows, we treat the general case without making special assumptions about *T*.

4.1. Sufficient and necessary conditions for problem feasibility

The following theorem provides clarity on the issue of problem feasibility.

Theorem 1: Math Model #1 has a feasible solution if and only if MaxTotalS ≥ MinTotalD.

Proof: We first show that the problem is infeasible if *MaxTotalS* < *MinTotalD*. If we sum up constraint (6) and all constraints of type (7) in Math Model #1, we arrive at the following:

$$\sum_{s=1}^{S}\sum_{t=1}^{T} SAmt_{s,t} = \sum_{d=1}^{D}\sum_{t=1}^{T} DAmt_{d,t} \tag{14}$$

In other words, the total amount supplied during the entire cycle should equal the total amount demanded. If *MaxTotalS* < *MinTotalD,* the supplies fall short of the demands; the above requirement cannot be met and the problem is infeasible. We next show that the problem is feasible if *MaxTotalS* ≥ *MinTotalD*. To do this, we observe that we can always construct a feasible solution whenever *MaxTotalS* ≥ *MinTotalD*. Section 4.2 presents a method for generating such a solution. ■

4.2. Method for automatically constructing a feasible solution

We now present a method for automatically generating a feasible solution to Math Model #1 whenever *MaxTotalS* ≥ *MinTotalD*. This method uses new concepts and terminology which we introduce here. Let us refer to an instance when units are taken from (delivered to) the buffer by a specific demander (supplier) at the beginning of a specific period as a *demand (supply) occurrence*.

We form a feasible solution by deciding the values of five *solution elements*: (i) demand start points, (ii) demand intervals, (iii) supply start points, (iv) supply intervals, and (v) supply subtraction epochs. A *demand start point* is a period at the beginning of which a batch of sufficient size is given to a particular demander; it serves as a reference point for all batches provided to this demander. *Demand intervals* indicate the times that elapse between consecutive demand occurrences for a given demander beginning with the demand start point. Consider the feasible solution shown in Table 2. In this solution, T5 could be demander 1's start point. If T5 is demander 1's start point, then (2, 3, 3, 2) are demander 1’s intervals because (T5, T7, T10, T3) are the demand occurrences for this demander; T5 and T7 are separated by 2 periods; T7 and T10 are separated by 3 periods; T10 and T3 are separated by 3 periods; and T3 and T5 are separated by 2 periods. Another possibility is to consider T3 as demander 1’s start point and (2, 2, 3, 3) as demander 1’s intervals. A *supply start point* is a period at the beginning of which a particular supplier delivers a batch to the buffer; it serves as a reference point for all batches delivered by this supplier. *Supply intervals* indicate the times that elapse between consecutive supply occurrences for a given supplier beginning with the supply start point. For instance, in Table 2, T1 could be supplier 3’s start point. If T1 is supplier 3’s start point, then (3, 4, 3) are supplier 3’s intervals. Note that each demander’s intervals and each supplier’s intervals should sum to *T*. Also, no demand interval for demander *d* should exceed DT_d, and no supply interval for supplier *s* should be less than ST_s. A *supply subtraction epoch* is a period when the amount delivered by a particular supplier *s* is less than SQ_s but greater than zero. For example, in Table 2, supplier 1 has two subtraction epochs—T7 and T9—where less than SQ_1 (= 4) is supplied. Supplier 2 has no supply subtraction epochs, and supplier 3 has three subtraction epochs—T1, T4, and T8—where less than SQ_3 (= 5) is supplied.

Our method decides the values of the above solution elements, and then uses these elements to construct a feasible solution to Math Model #1. The method is summarized in Table 6. We use the instance shown in Table 7 to illustrate this method. In this instance, $MaxTotalS = 80 \geq 79 = MinTotalD$.

In step 1, we randomly generate demand and supply occurrences that agree with the demand and supply timing requirements (DT_d and ST_s) and quantities (DQ_d and SQ_s) so as to satisfy constraints (2-5) in Math Model #1. First, a random demand start point from 1 to *T* is selected for each demander. Then, random demand intervals are generated for each demander so as to agree with constraint (2). In particular,

Table 6. Summary of procedure for automatically generating a feasible solution

Step	Explanation
1	Generate random demand and supply occurrences with full batch sizes
2	Reduce supplies (i.e. decide supply subtraction epochs)
3	Build initial inventory diagram
4	Move X-axis vertically so the lowest inventory level equals 0
5	Move Y-axis horizontally so the inventory level in period *T* is 0

Table 7. Input parameters for Illustrative Instance #2

Number of demanders: 6		Number of suppliers: 6	
DT_1: 3 ($MinNumD_1$ = 6)	DQ_1: 2	ST_1: 9 ($MaxNumS_1$ = 1)	SQ_1: 7
DT_2: 7 ($MinNumD_2$ = 3)	DQ_2: 4	ST_2: 8 ($MaxNumS_2$ = 2)	SQ_2: 6
DT_3: 6 ($MinNumD_3$ = 3)	DQ_3: 2	ST_3: 6 ($MaxNumS_3$ = 2)	SQ_3: 9
DT_4: 5 ($MinNumD_4$ = 4)	DQ_4: 2	ST_4: 7 ($MaxNumS_4$ = 2)	SQ_4: 4
DT_5: 3 ($MinNumD_5$ = 6)	DQ_5: 1	ST_5: 5 ($MaxNumS_5$ = 3)	SQ_5: 9
DT_6: 4 ($MinNumD_6$ = 5)	DQ_6: 7	ST_6: 7 ($MaxNumS_6$ = 2)	SQ_6: 4
T = 17			

for each *d*, we let demander *d*'s demand intervals be a set of $MinNumD_d$ random positive integers that sum to *T*, each of which is ≤ DT_d. This is done by initially setting each interval to DT_d and then repeatedly reducing by one the value of a randomly selected interval until the intervals sum to *T*. The likelihood of an interval next being selected for reduction is proportional to its current value. To satisfy constraint (3), the amount demanded by demander *d* for each of his/her demand occurrences is set equal to DQ_d for all *d*.

Next, a random supply start point from 1 to *T* is selected for each supplier. Then, random supply intervals are generated for each supplier so as to agree with constraint (4). In particular, for each *s*, we let supplier *s*'s supply intervals be a set of $MaxNumS_s$ random positive integers that sum to *T*, each of which is ≥ ST_s. This is done by initially setting each interval to ST_s and then repeatedly increasing by one the value of a randomly selected interval until they sum to *T*. Each interval is equally likely to next be increased. To satisfy constraint (5), the amount supplied by supplier *s* for each of his/her supply occurrences is initially set equal to SQ_s for all *s*.

Figure 2 shows one possible result of the above process applied to Illustrative Instance #2. We

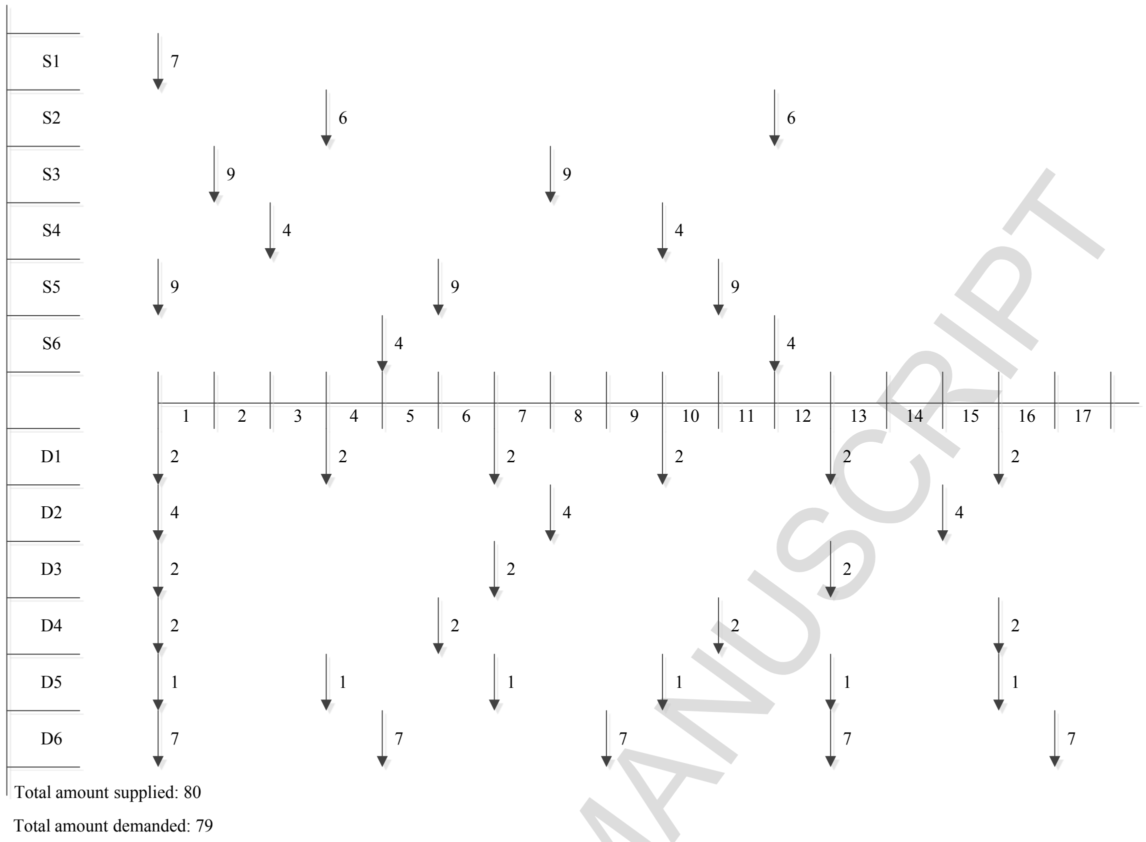


Figure 2. Step 1 in procedure for constructing a feasible solution: generate random demands and supplies

call this an *initial supply and demand diagram*. Note that the total amount supplied (demanded) is 80 (79) units per cycle. Thus, condition (14) is not satisfied.

In step 2, we reduce some of the supply amounts until the total amount supplied during the cycle equals the total amount demanded. In other words, we decide the supply subtraction epochs (i.e. places where the amount supplied is less than a supplier’s ability to supply). To do this, we keep randomly deleting one unit of supply until the total amount supplied during the cycle equals the total amount demanded. Each unit of supply in the initial supply and demand diagram is equally likely to be the next one deleted. Then we obtain a *balanced supply and demand diagram* as shown in Figure 3. In this figure, the bold number and arrow show that the size of one of supplier 1’s batches has been reduced. The bottom of this figure also shows the total amount supplied and demanded in each period. In this diagram, condition (14) is satisfied.

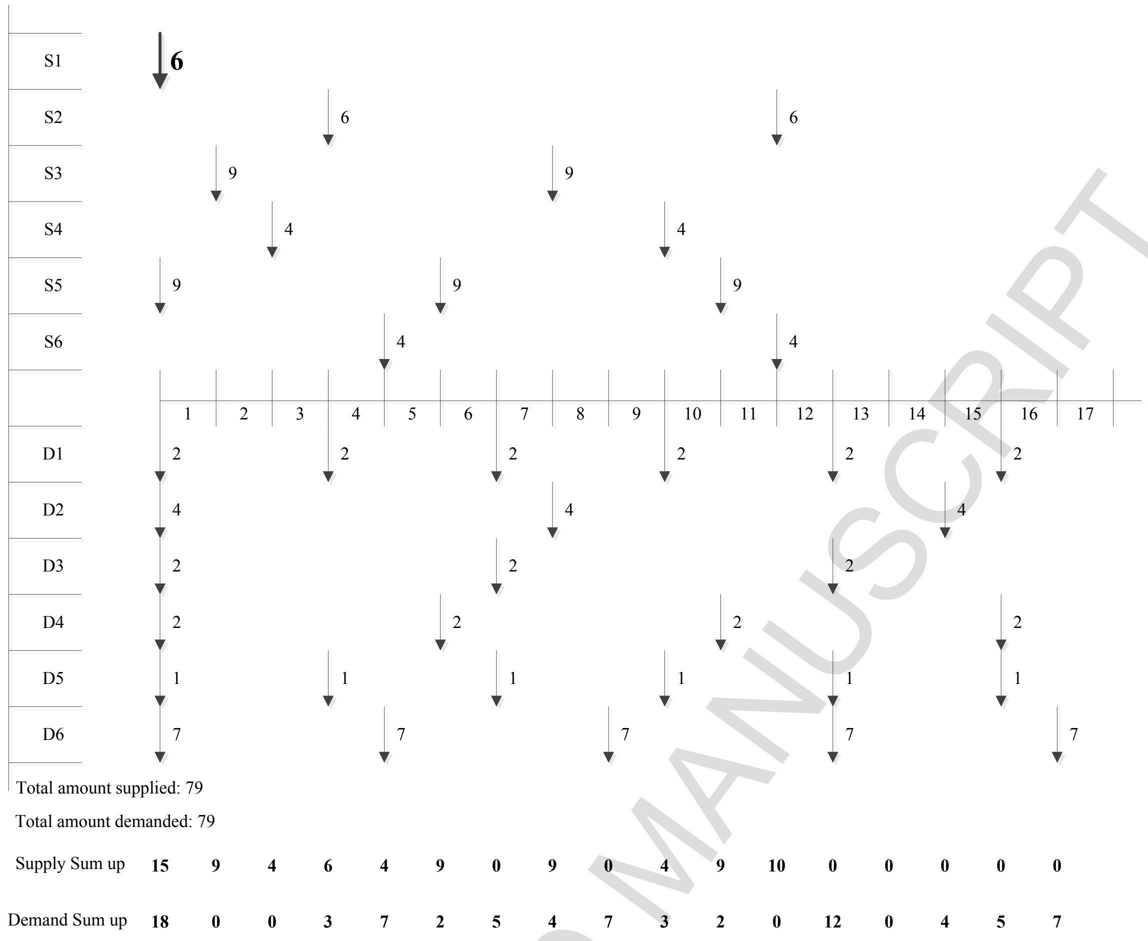

S1
6
S2
6
6
S3
9
9
S4
4
4
S5
9
9
9
S6
4
4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
D1
2 2 2 2 2 2
D2
4 4 4
D3
2 2 2
D4
2 2 2 2
D5
1 1 1 1 1 1
D6
7 7 7 7 7
Total amount supplied: 79
Total amount demanded: 79
Supply Sum up 15 9 4 6 4 9 0 9 0 4 9 10 0 0 0 0 0
Demand Sum up 18 0 0 3 7 2 5 4 7 3 2 0 12 0 4 5 7

Figure 3. Step 2 in procedure for constructing a feasible solution: reduce supplies

In step 3, we first compute the *net amount supplied* (amount supplied minus amount demanded) during each period. This is displayed in the “Balance” row in Figure 4. Then, assuming the inventory level is zero immediately prior to the start of the first period, we compute the inventory level during each period in the cycle. This is displayed in the “Inventory” row in Figure 4. Then, we draw an *initial inventory diagram* that shows the inventory level over the entire cycle (Figure 4). This diagram satisfies constraints (2-7) but may violate the non-negativity requirement for I_t.

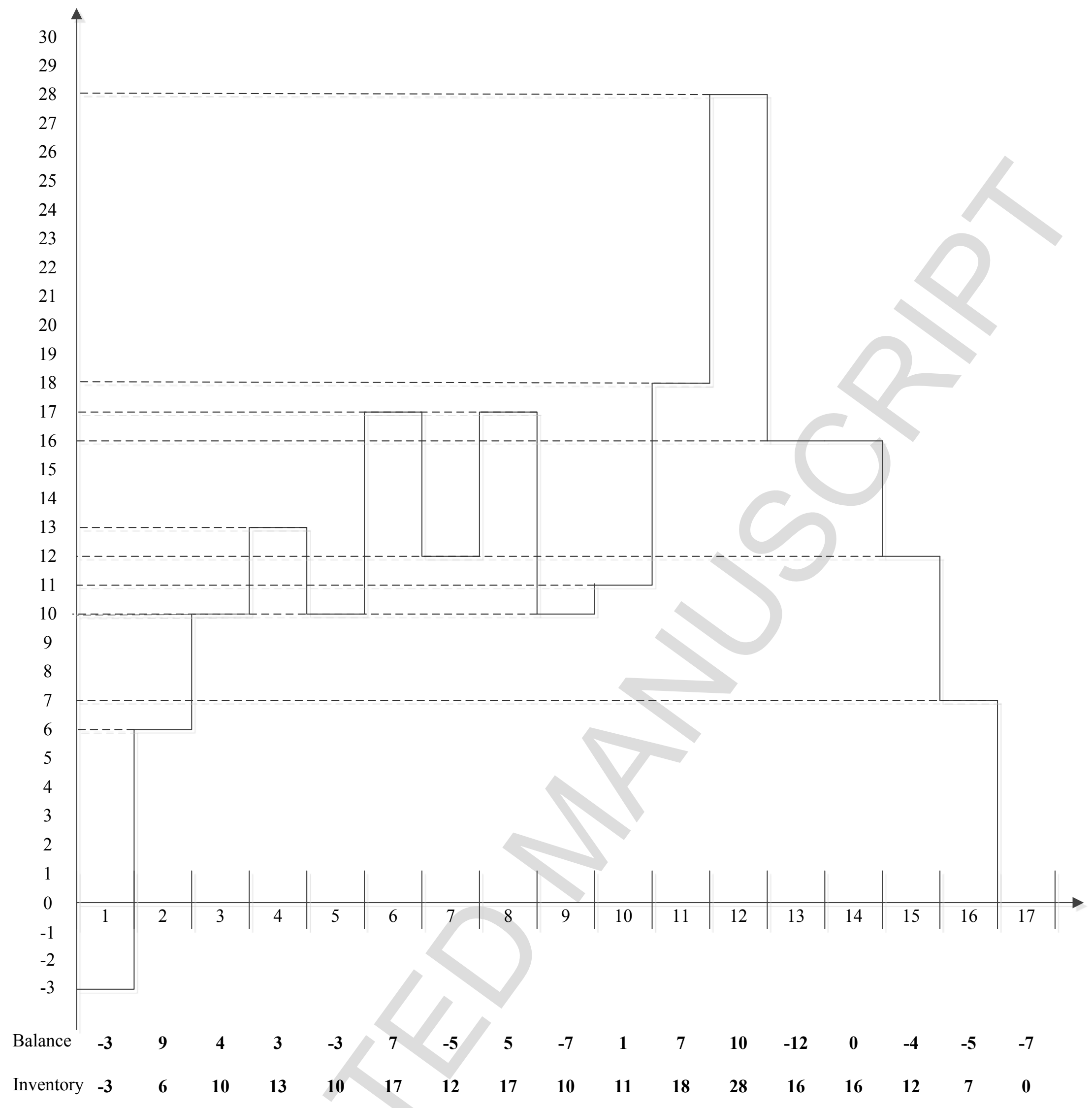


Figure 4. Step 3 in procedure for constructing a feasible solution: build initial inventory diagram

In step 4, we compute the lowest inventory level observed during the cycle and subtract this from every inventory value in the cycle. The resulting inventory diagram, shown in Figure 5, satisfies constraints (2-7) and the non-negativity requirement for I_t but will likely violate constraint (9).

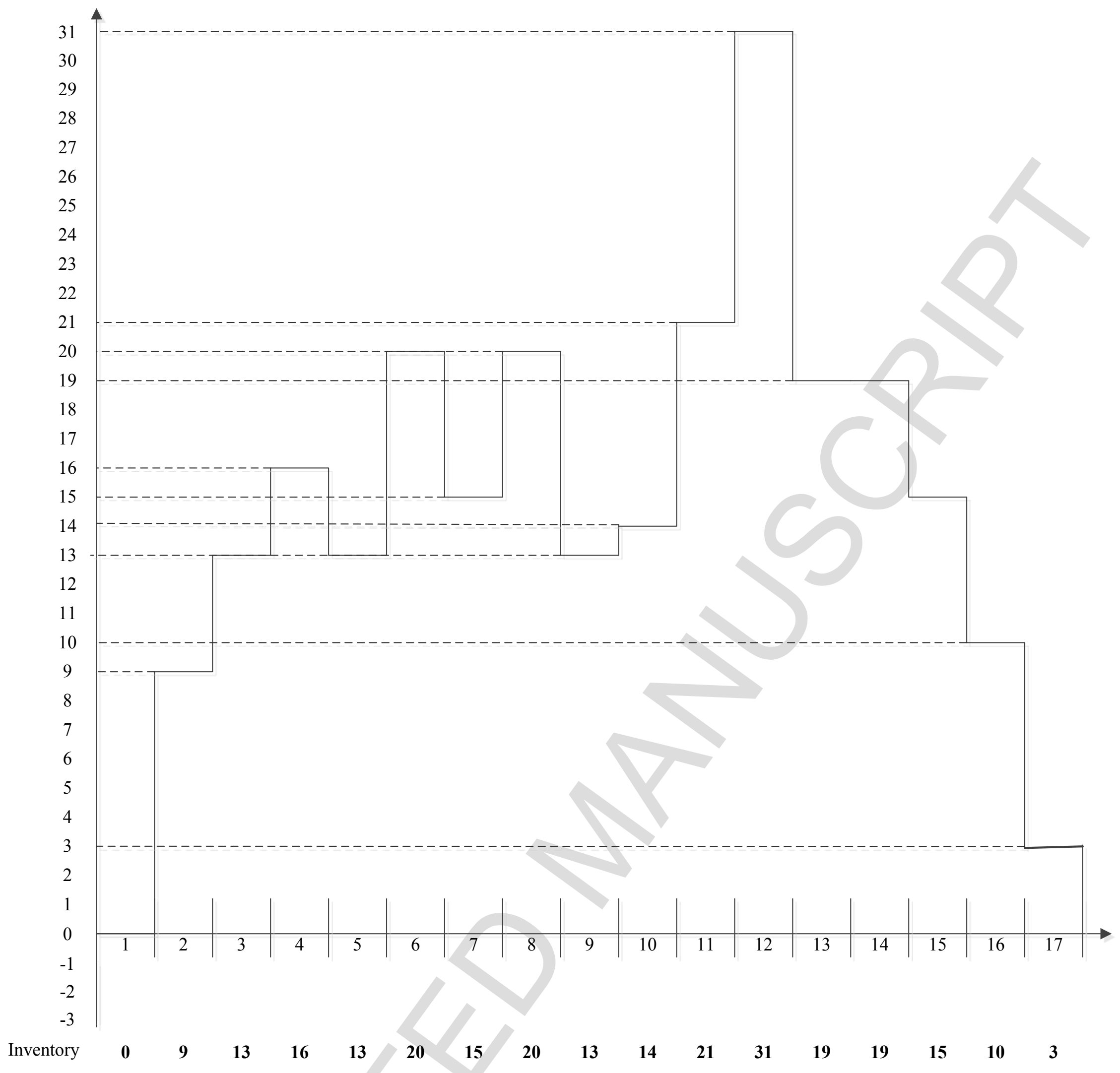
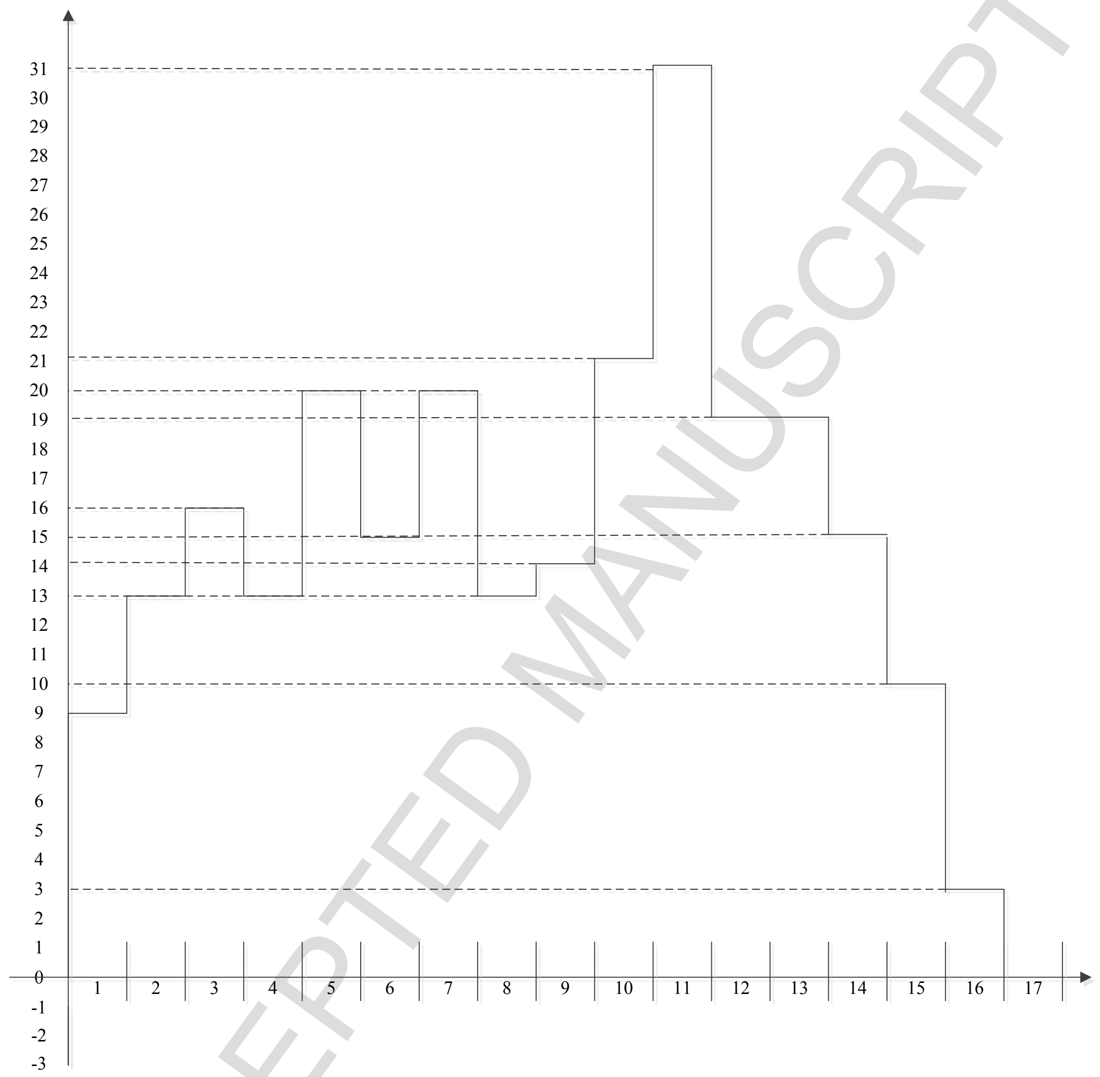


Figure 5. Step 4 in procedure for constructing a feasible solution: move x-axis up or down

In step 5, we shift the Y-axis of the inventory diagram—changing the period that appears first and wrapping all inventory values around the diagram in a cyclic manner—until the inventory level during period *T* is zero. Figure 6 shows the resulting *final inventory diagram* that depicts a feasible solution to Math Model #1. Then we look at the inventory levels and compute the two objective values. In this example, the value of objective 1 (2) is 251 (31) units. Finally, we recall the times and amounts of all supply and demand occurrences. This information—shown near the bottom of Figure 6—fully defines a feasible solution to the problem instance.

Note that the above five-step procedure can be repeated to randomly generate a variety of feasible solutions to the same problem instance. This procedure is a subroutine within four of the five solution methods considered in Section 6.



Inventory	**9**	**13**	**16**	**13**	**20**	**15**	**20**	**13**	**14**	**21**	**31**	**19**	**19**	**15**	**10**	**3**	**0**
Supplier 1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**6**
Supplier 2	-	-	**6**	-	-	-	-	-	-	-	**6**	-	-	-	-	-	-
Supplier 3	**9**	-	-	-	-	-	**9**	-	-	-	-	-	-	-	-	-	-
Supplier 4	-	**4**	-	-	-	-	-	-	**4**	-	-	-	-	-	-	-	-
Supplier 5	-	-	-	-	**9**	-	-	-	-	**9**	-	-	-	-	-	-	**9**
Supplier 6	-	-	-	**4**	-	-	-	-	-	-	**4**	-	-	-	-	-	-
Demander 1	-	-	**2**	-	-	**2**	-	-	**2**	-	-	**2**	-	-	**2**	-	**2**
Demander 2	-	-	-	-	-	-	**4**	-	-	-	-	-	-	**4**	-	-	**4**
Demander 3	-	-	-	-	-	**2**	-	-	-	-	-	**2**	-	-	-	-	**2**
Demander 4	-	-	-	-	**2**	-	-	-	-	**2**	-	-	-	-	**2**	-	**2**
Demander 5	-	-	**1**	-	-	**1**	-	-	**1**	-	-	**1**	-	-	**1**	-	**1**
Demander 6	-	-	-	**7**	-	-	-	**7**	-	-	-	**7**	-	-	-	**7**	**7**
DI	**0**	**0**	**3**	**7**	**2**	**5**	**4**	**7**	**3**	**2**	**0**	**12**	**0**	**4**	**5**	**7**	**18**

Figure 6. Step 5 in procedure for constructing a feasible solution: move y-axis left or right

4.3. Two observations concerning solution optimality

Math Model #1 can be tightened to allow optimal solutions to be obtained in the same or less time. The following theorem provides the basis for this tightening.

Theorem 2: There always exists an optimal solution to Math Model #1 in which $DAmt_{d,t}$ equals either 0 or DQ_d for all d and all t. In other words, there exists an optimal solution in which the demand batch sizes are exactly met and never exceeded.

Proof: We show it would be absurd for either (i) $0 < DAmt_{d,t} < DQ_d$ or (ii) $DAmt_{d,t} > DQ_d$ for any (d, t). Note that, in both cases (i) and (ii), the "extra units of demand" are not helping to satisfy any constraints in Math Model #1 beyond what the values (i) $DAmt_{d,t}$ = 0 and (ii) $DAmt_{d,t} = DQ_d$ accomplish respectively. Consider any feasible solution Z in which one or more "extra units of demand" in the form of (i) or (ii) exist. From this solution, we can generate another solution Z' in which $DAmt_{d,t}$ = 0 or DQ_d for all (d, t) such that the values of objectives 1 and 2 for Z' are both less than or equal to their values for Z. To do this, consider each "extra unit of demand" one at a time in solution Z. Delete each such extra unit of demand, and delete one unit of supply that occurs in the same period (if possible) or the period that is earlier than and as close as possible to this period in the cyclic sense. The resulting solution Z' (a) has objectives 1 and 2 no higher than Z and (b) is feasible owing to the feasibility of Z and the non-negativity of the inventory values in Z.

Table 8 shows an example of the above process. The top half of the table shows a feasible solution Z for Illustrative Instance #1 (Table 1) with one "extra unit of demand" that is highlighted. The value of objective 1 (2) for this solution is 63 (9). The bottom half of the table shows feasible solution Z' for this instance that is obtained using the above process. The highlighted values have been changed. The value of objective 1 (2) for solution Z' is 62 (9), which is less than or equal to the respective value for Z. ■

Table 8. Example supporting the proof of Theorem 2

Z	Period									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Demander 1	0	0	2	0	2	0	2	0	0	2
Demander 2	0	4	0	4	0	**5**	0	4	0	4
Demander 3	0	0	0	2	0	0	0	0	0	2
Sum up (DI)	0	4	2	6	2	5	2	4	0	8
Supplier 1	4	0	4	0	4	0	1	0	3	0
Supplier 2	0	3	0	0	0	0	3	0	0	0
Supplier 3	4	0	0	4	0	0	0	3	0	0
Sum up (SI)	8	3	4	4	4	0	4	3	3	0
SI-DI	8	-1	2	-2	2	-5	2	-1	3	-8
Inventory held	8	7	9	7	9	4	6	5	8	0
Objective 1 (Total inventory)	63	Objective 2 (Maximum inventory)	9							
Z'	Period									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Demander 1	0	0	2	0	2	0	2	0	0	2
Demander 2	0	4	0	4	0	**4**	0	4	0	4
Demander 3	0	0	0	2	0	0	0	0	0	2
Sum up (DI)	0	4	2	6	2	**4**	2	4	0	8
Supplier 1	4	0	4	0	**3**	0	1	0	3	0
Supplier 2	0	3	0	0	0	0	3	0	0	0
Supplier 3	4	0	0	4	0	0	0	3	0	0
Sum up (SI)	8	3	4	4	**3**	0	4	3	3	0
SI-DI	8	-1	2	-2	**1**	**-4**	2	-1	3	-8
Inventory held	8	7	9	7	**8**	4	6	5	8	0
Objective 1 (Total inventory)	**62**	Objective 2 (Maximum inventory)	9							

We now discuss the diversity of optimal solutions to Math Model #1.

Observation 1: There exist problem instances of the CICP-MFBSDP in which all optimal solutions have one or more "extra demand occurrences," i.e. one or more d such that the number of demand occurrences for demander d is strictly greater than MinNumD$_d$ (see equation 10).

Discussion: Table 9 shows an example of such a problem instance which has two suppliers of equal capability and one demander. An optimal solution to this instance is shown in Table 10; the optimal values of objectives 1 and 2 are both 0. In other words, this instance has a *zero inventory solution*. Note in this optimal solution that four batches—more than the minimum number of batches = 3 =

MinNumD$_1$—are given to the demander. Furthermore, there is no feasible solution in which the objective values are 0 and only three batches are provided to the demander. Indeed, if only three batches were provided to the demander, one batch would be provided every four periods exactly, e.g. in periods T1, T5, and T9. To have a zero inventory solution, the suppliers would have to deliver exactly three batches during these periods; one supplier would deliver one batch and one supplier would deliver two batches. Without loss of generality (owing to the cyclic nature of the problem), assume that supplier 1 delivers one batch in period T1 and supplier 2 delivers two batches. But the two batches delivered by supplier 2 could not possibly occur during periods T5 and T9 because $ST_2 = 6$. Thus, there is no zero inventory solution in which only three batches are provided to the demander. ■

Table 9. Input parameters for Illustrative Instance #3

Number of demanders: 1		Number of suppliers: 2	
DT_1: 4 ($MinNumD_1 = 3$)	DQ_1: 2	ST_1: 6 ($MaxNumS_1 = 2$)	SQ_1: 2
$T = 12$		ST_2: 6 ($MaxNumS_1 = 2$)	SQ_2: 2

Table 10. Optimal solution for Illustrative Instance #3

	Period											
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Demander 1 (DI)	2	0	0	2	0	0	2	0	0	2	0	0
Supplier 1	2	0	0	0	0	0	2	0	0	0	0	0
Supplier 2	0	0	0	2	0	0	0	0	0	2	0	0
Sum up (SI)	2	0	0	2	0	0	2	0	0	2	0	0
SI-DI	0	0	0	0	0	0	0	0	0	0	0	0
Inventory held	0	0	0	0	0	0	0	0	0	0	0	0
Objective 1 (Total inventory)	0	Objective 2 (Maximum inventory)	0									

Observation 1 shows that the set of optimal solutions to instances of Math Model #1 is more diverse than we might expect. Indeed, there are problem instances in which extra demand occurrences are needed to optimize the coordination between demands and supplies. In these instances, we observe *higher throughput with lower inventories* when extra demand occurrences are present. This result suggests that while allowing extra demand occurrences may be attractive in terms of inventory reduction, it may lead to undesirable performance in other aspects. Indeed, in the context of the CICP-MFBSDP, it may lead to a linearly increasing number of products being wasted at the demander stage over time. Hence, it is important to account for this risk in the problem setup. In what follows, we do so by modifying the original setup of the CICP-MFBSDP with one additional assumption to remove solutions with extra demand occurrences—such as that shown in Table 10—from consideration.

Assumption 1: We only consider the feasible solutions to Math Model #1 in which there are no "extra demand occurrences," i.e. in which the number of demand occurrences for demander d equals $MinNumD_d$ for all d (see equation 10).

Assumption 1 requires us to distinguish two variations of the problem. Let the abbreviation CICP-MFBSDP-1 refer to the problem at hand with assumption 1. Let CICP-MFBSDP-2 refer to the problem without assumption 1. *The remainder of this paper shall exclusively focus on CICP-MFBSDP-1.*

4.4. Tight mathematical formulation of CICP-MFBSDP-1

We now reformulate Math Model #1 to incorporate Assumption 1 and some tighter constraints based on Theorem 2. Theorem 2 allows us to tighten constraint (3) in Math Model #1 by changing the inequality to an equality. To incorporate Assumption 1, we add new constraints (15-16) which require that the total amount demanded and total amount supplied during the cycle equal the minimum possible value for each—*MinTotalD* (see equation 11). The resulting model—Math Model #2—is a valid formulation of problem CICP-MFBSDP-1 but not problem CICP-MFBSDP-2.

In Math Model #2, constraint (3') specifies that the batch size provided to demander d should equal 0 or DQ_d in all cases. Constraints (15-16) ensure that the total amount demanded and supplied during the cycle equal *MinTotalD*. *The remainder of this paper exclusively focuses on Math Model #2.*

Math Model #2:

(1), (2), (4), (5), (6), (7), (8), (9) from Math Model #1

$$DAmt_{d,t} = (DQ_d)(DYN_{d,t}) \qquad \forall d \; \forall t \tag{3'}$$

$$\sum_{d=1}^{D}\sum_{t=1}^{T} DAmt_{d,t} = MinTotalD \tag{15}$$

$$\sum_{s=1}^{S}\sum_{t=1}^{T} SAmt_{s,t} = MinTotalD \tag{16}$$

5. Three heuristic algorithms

The single item CLSP is NP hard (Florian et al. 1980, Bitran and Yanasse 1982), so we expect it is necessary to develop efficient algorithms for solving large instances of Math Model #2. In this section, three heuristic methods are developed for serving this goal. The first is a "random" algorithm that repeatedly generates random feasible solutions—according to the method from Section 4.2—and reports the best solution found within a predefined time limit. This algorithm provides a benchmark for other methods. The second method is a simulated annealing (SA) heuristic, and the third method is a genetic algorithm (GA).

5.1. Simulated annealing heuristic

The procedure of our SA algorithm is shown in Figure 7. An initial feasible solution is generated by the method described in Section 4.2. This solution is entirely specified by five elements: its (1) demand start points, (2) demand intervals, (3) supply start points, (4) supply intervals, and (5) supply subtraction epochs (see Section 4.2).

Five neighborhoods corresponding to the above five solution elements are used to generate neighboring solutions. In the first neighborhood structure, a random number from 1 to *D* of demand start points are changed to new random values between 1 and *T*. In the second neighborhood structure, the demand intervals are changed to new, random values for a random number of demanders from 1 to *D*. In the third neighborhood structure, a random number from 1 to *S* of supply start points are changed to new random values between 1 and *T*. In the fourth neighborhood structure, the supply intervals are changed to

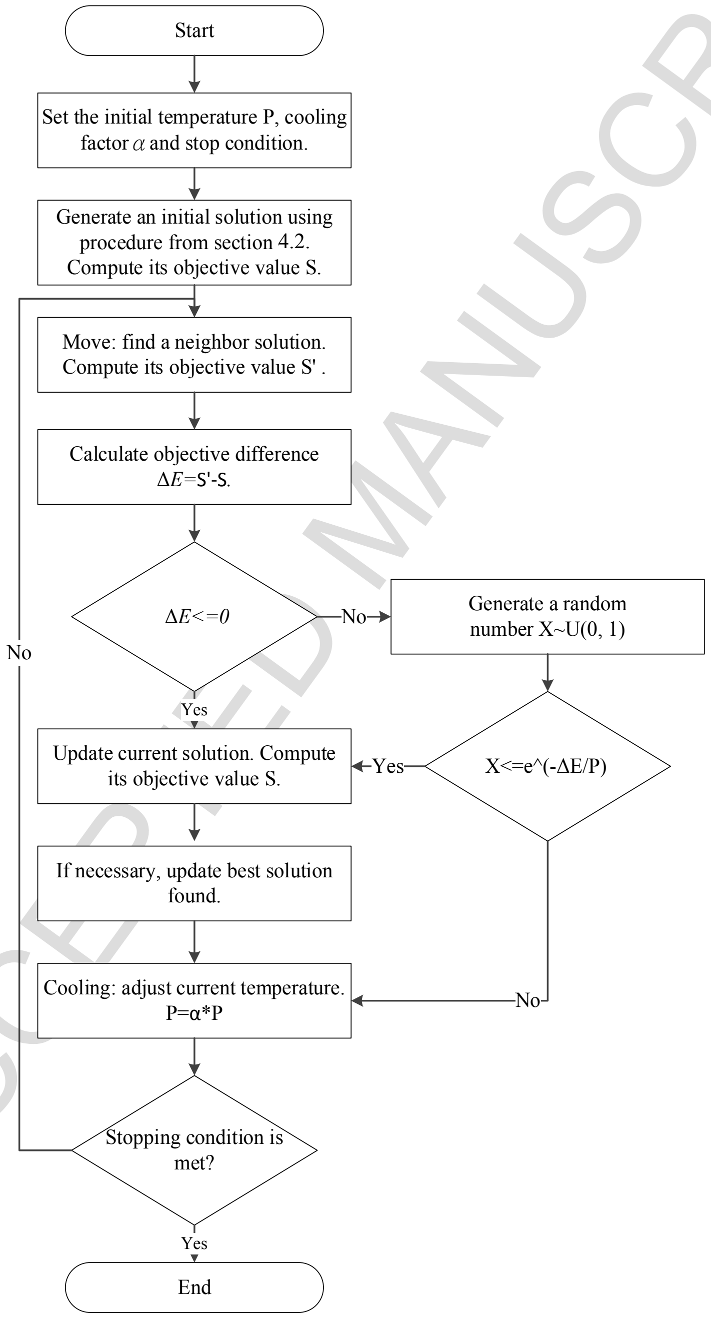

Start
Set the initial temperature P, cooling factor α and stop condition.
Generate an initial solution using procedure from section 4.2. Compute its objective value S.
Move: find a neighbor solution. Compute its objective value S' .
Calculate objective difference ΔE=S'-S.
ΔE<=0
No
Generate a random number X~U(0, 1)
Yes
X<=e^(-ΔE/P)
Yes
Update current solution. Compute its objective value S.
If necessary, update best solution found.
No
Cooling: adjust current temperature. P=α*P
No
Stopping condition is met?
Yes
End

Figure 7. Simulated annealing algorithm procedure

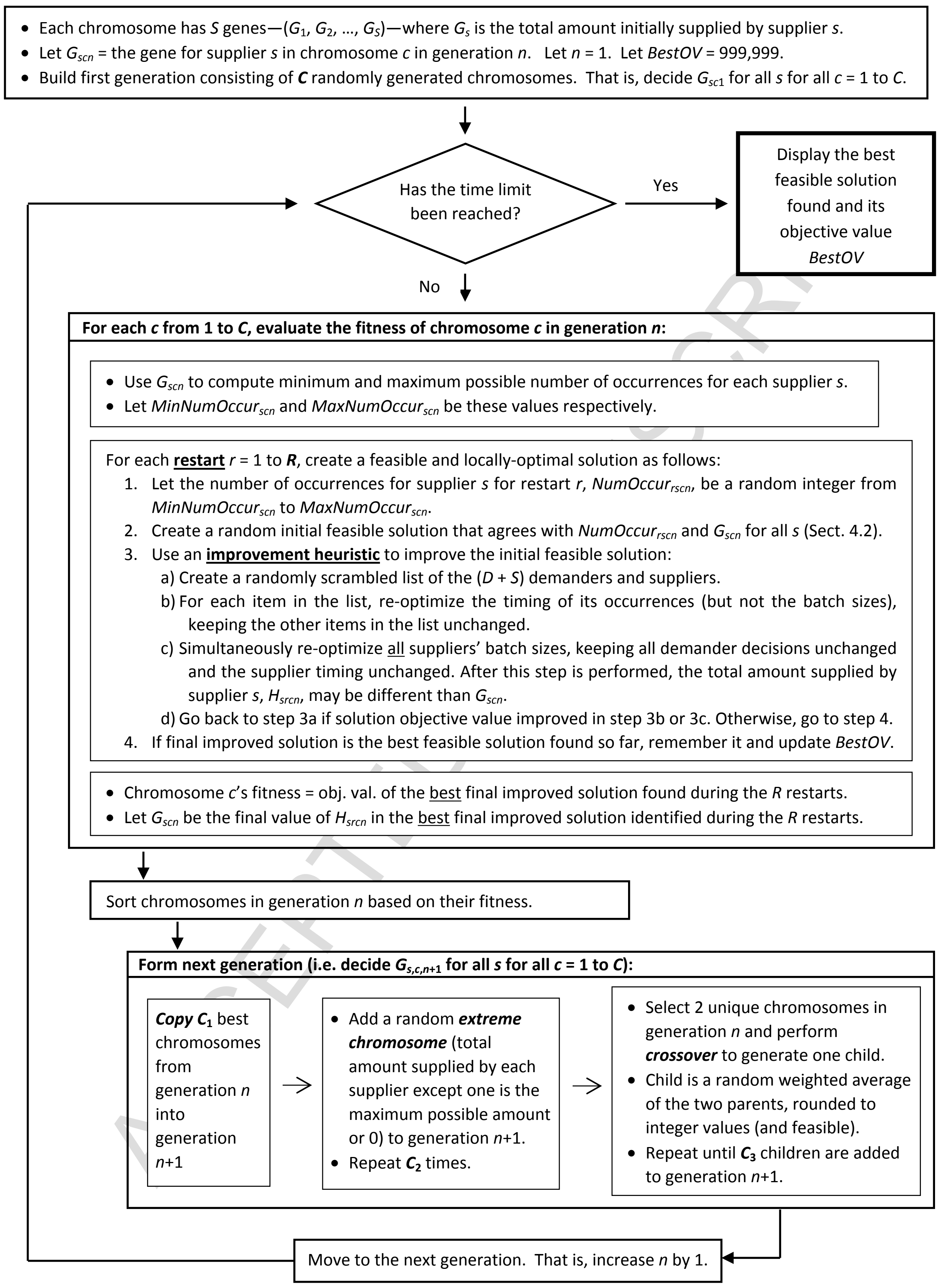
new, random values for a random number of suppliers from 1 to S. In the fifth neighborhood structure, the supply subtraction epochs are changed to new randomly selected values. The probability of using neighborhood 5 is min{2X, 20}/100 where $X = MaxTotalS - MinTotalD$. This ranges from 0% to 20%. The probability of using each of the neighborhoods 1-4 is (1 - probability of using neighborhood 5)/4.

The procedure from Section 4.2 is used to construct neighboring solutions based on the new values for the five solution elements. The neighboring solution automatically replaces the current solution if its objective value is less than or equal to that of the current solution; otherwise, it replaces the current solution with a probability $e^{-\Delta E/P}$ where ΔE is the difference between the corresponding objective values and P is the temperature which is gradually decreased during the process by a cooling factor α. Computations cease when a predetermined time limit is reached.

5.2. Genetic algorithm

In our GA, each feasible solution is represented by a *chromosome* consisting of a sequence of S *genes*—(G_1, G_2, G_3, …, G_S)—where G_s is the gene for supplier s. The gene for supplier s specifies the total amount supplied by supplier s in the initial feasible solutions that are created for that chromosome. For example, the feasible solution generated in Section 4.2 is one of many initial feasible solutions for chromosome (6, 12, 18, 8, 27, 8), i.e. in which a total of (6, 12, 18, 8, 27, 8) units are supplied by supplier (1, 2, 3, 4, 5, 6) respectively. Another possible chromosome for Illustrative Instance #2 (Table 7) is (7, 11, 18, 8, 27, 8). Note that the genes in a chromosome must sum to $MinTotalD$ and satisfy $0 \leq G_s \leq (MaxNumS_s)(SQ_s)$ for all s.

Figure 8 shows the overall GA procedure. The first step is to randomly construct the C chromosomes which comprise the first generation. That is, for each c from 1 to C, we decide G_s for all s from 1 to S. This is done by initially setting $G_s = (MaxNumS_s)(SQ_s)$ and then repeatedly reducing by one the value of a randomly selected gene until the gene values sum to $MinTotalD$. The likelihood of a gene next being selected for value reduction is proportional to its current value.



- Each chromosome has S genes—(G_1, G_2, …, G_S)—where G_s is the total amount initially supplied by supplier s.
- Let G_{scn} = the gene for supplier s in chromosome c in generation n. Let n = 1. Let *BestOV* = 999,999.
- Build first generation consisting of $\boldsymbol{C}$ randomly generated chromosomes. That is, decide G_{sc1} for all s for all c = 1 to C.

Has the time limit been reached?

Yes

Display the best feasible solution found and its objective value *BestOV*

No

For each c from 1 to C, evaluate the fitness of chromosome c in generation n:

- Use G_{scn} to compute minimum and maximum possible number of occurrences for each supplier s.
- Let $MinNumOccur_{scn}$ and $MaxNumOccur_{scn}$ be these values respectively.

For each **restart** r = 1 to $\boldsymbol{R}$, create a feasible and locally-optimal solution as follows:

1. Let the number of occurrences for supplier s for restart r, $NumOccur_{rscn}$, be a random integer from $MinNumOccur_{scn}$ to $MaxNumOccur_{scn}$.
2. Create a random initial feasible solution that agrees with $NumOccur_{rscn}$ and G_{scn} for all s (Sect. 4.2).
3. Use an **improvement heuristic** to improve the initial feasible solution:
   a) Create a randomly scrambled list of the (D + S) demanders and suppliers.
   b) For each item in the list, re-optimize the timing of its occurrences (but not the batch sizes), keeping the other items in the list unchanged.
   c) Simultaneously re-optimize all suppliers' batch sizes, keeping all demander decisions unchanged and the supplier timing unchanged. After this step is performed, the total amount supplied by supplier s, H_{srcn}, may be different than G_{scn}.
   d) Go back to step 3a if solution objective value improved in step 3b or 3c. Otherwise, go to step 4.
4. If final improved solution is the best feasible solution found so far, remember it and update *BestOV*.

- Chromosome c's fitness = obj. val. of the best final improved solution found during the R restarts.
- Let G_{scn} be the final value of H_{srcn} in the best final improved solution identified during the R restarts.

Sort chromosomes in generation n based on their fitness.

Form next generation (i.e. decide $G_{s,c,n+1}$ for all s for all c = 1 to C):

Copy C_1 best chromosomes from generation n into generation n+1

- Add a random ***extreme chromosome*** (total amount supplied by each supplier except one is the maximum possible amount or 0) to generation n+1.
- Repeat $\boldsymbol{C_2}$ times.

- Select 2 unique chromosomes in generation n and perform ***crossover*** to generate one child.
- Child is a random weighted average of the two parents, rounded to integer values (and feasible).
- Repeat until $\boldsymbol{C_3}$ children are added to generation n+1.

Move to the next generation. That is, increase n by 1.

Figure 8. Genetic algorithm procedure.

5.2.1. Chromosome fitness computation

The iterative portion of the GA commences immediately after the first generation of chromosomes is formed. In the first half of each iteration, the *fitness* (i.e. objective value) of each chromosome in the current generation is evaluated by (i) using the method from Section 4.2 to create an initial feasible solution that agrees with its genes, (ii) improving this solution using simple local search (i.e. an *improvement heuristic*) until a local optimal solution is identified, (iii) computing the objective value of the final improved solution, and (iv) repeating steps i-iii R times and remembering the objective value of the best final improved solution that was identified during the R restarts.

We now elaborate on items i-iv above. Regarding item (i), each time a restart occurs we first decide the number of occurrences for each supplier s, $NumOccur_s$. This is a random integer in the range [$MinNumOccur_s$, $MaxNumOccur_s$] where the former equals G_s/SQ_s rounded up to the nearest integer and the latter equals the smaller of G_s and $MaxNumS_s$. Then, we use the method from Section 4.2 to create an initial feasible solution that agrees with $NumOccur_s$ and G_s for all s. In particular, for each s we let supplier s's intervals be a set of $NumOccur_s$ random positive integers that sum to T, each of which is $\geq ST_s$ (Section 4.2). Then we initially set the batch size for each of supplier s's occurrences to SQ_s, and we repeatedly reduce the batch size of a randomly selected occurrence for supplier s by one until the total amount he/she supplies is G_s. The likelihood of an occurrence next being selected for batch size reduction is proportional to its current batch size.

Regarding (ii), we consider two types of solution improvements—(1) re-optimizing the timing of the occurrences (but not the batch sizes) for a given supplier or demander while keeping the decisions for all other suppliers and demanders unchanged and (2) simultaneously re-optimizing the batch sizes for all suppliers while keeping all other solution elements unchanged. These improvements are easy to implement and do not require sophisticated math programming techniques.

The type 1 improvements are accomplished by pushing supply occurrences to the right (i.e. scheduling supply occurrences later) or pushing demand occurrences to the left (i.e. scheduling demand occurrences earlier). For example, the feasible solution in Figure 6 can be improved by re-optimizing the timing of supplier 2's occurrences which currently take place at the beginning of periods 3 and 11. If each occurrence takes place one period later, the inventory level in periods 3 and 11 is reduced by six and the inventory level in other periods is unchanged. If each occurrence takes place two periods later, the inventory level in periods 3, 4, 11, and 12 is reduced by six. Continuing this train of thought, we see it is possible to push each of supplier 2's occurrences to the right by five periods (but no more than that) while maintaining feasibility, i.e. while still respecting ST_2 and the non-negative inventory requirement. The resulting solution has six fewer units of inventory during periods 3, 4, 5, 6, 7, 11, 12, 13, 14, and 15, and it has value 191 (25) for objective 1 (2); this is the best we can do when re-optimizing the timing of supplier 2's occurrences. The solution in Figure 6 can also be improved by re-optimizing the timing of demander 1's occurrences which currently take place at the beginning of periods 3, 6, 9, 12, 15, and 17. If each occurrence takes place one period earlier, the inventory level in six periods is reduced by two. In short, the best way to re-optimize demander 1's occurrences is to push them further left until they take place at the beginning of periods 1, 3, 6, 9, 12, and 15 respectively. The resulting solution is feasible—it respects DT_1 and the non-negative inventory requirement—and has two fewer units of inventory during periods 1-16; it has value 219 (29) for objective 1 (2) which is a nontrivial improvement. Overall, it is easy to imagine a procedure which automates the pushing of occurrences for a given supplier (demander) as far as possible to the right (left) while still maintaining a feasible solution.

The type 2 improvement re-optimizes the supply amounts $SAmt_{s,t}$ for all (s, t) such that $SYN_{s,t} = 1$. (The supply amounts $SAmt_{s,t}$ for all (s, t) such that $SYN_{s,t} = 0$ remain unchanged and equal to zero.) We begin by (a) resetting to zero the supply amounts $SAmt_{s,t}$ for all (s, t) such that $SYN_{s,t} = 1$ and (b) computing DI_t = the total amount demanded at the beginning of period t for each t. The main idea of the re-optimization process is to (c) satisfy the demand from the latest period u whose demand DI_u remains unsatisfied by repeatedly increasing the $SAmt_{s,t}$ values with the highest t such that $t \leq u$, $SYN_{s,t} = 1$, and $SAmt_{s,t} < SQ_s$. This process continues until the demand amounts DI_u from all periods have been satisfied.

We now show how the type 2 improvement process improves the solution shown in Figure 6. In part a, we reset the twelve positive $SAmt_{s,t}$ values to zero; these are the only $SAmt_{s,t}$ values that may later be increased. The computations for part b are shown at the bottom of Figure 6. In part c, we first focus on satisfying the 18 units of demand that occur at the beginning of period 17. This is done by letting ($SAmt_{1,17}$, $SAmt_{5,17}$, $SAmt_{2,11}$) = (7, 9, 2) respectively. Note that these values must not exceed (7, 9, 6) = (SQ_1, SQ_5, SQ_2) respectively. Then we focus on satisfying the 7 units of demand that occur at the beginning of period 16. This is done by changing the values of ($SAmt_{2,11}$, $SAmt_{6,11}$) from (2, 0) to (6, 3) respectively. We then consider how best to satisfy the demands that occur in periods 15, 14, 12, 10, and so on. The process terminates after the DI_3 = 3 units of demand in period 3 have been satisfied. Overall, we attempt to supply product to the buffer at times that are as close as possible to, but no later than, the times when it is demanded from the buffer. Clearly there is no better way to re-optimize the supplier batch sizes while keeping the demander decisions and supplier timing unchanged. The final, net result of this process for Figure 6 is that ($SAmt_{1,17}$, $SAmt_{3,1}$) will change from (6, 9) to (7, 8) respectively. The resulting solution has one fewer unit of inventory during periods 1-16 and has value 235 (30) for objective 1 (2). The improvement in this case is modest because $MaxTotalS$ = 80 is only slightly greater than $MinTotalD$ = 79, so there is not much freedom to modify the supplier batch sizes.

The aforementioned two types of solution improvements are carried out in the following sequence. First, the (S + D) possible type 1 improvements—one for each supplier and demander—are attempted in random order. Next, the type 2 improvement is carried out. If at least one of the (S + D + 1) improvement attempts yields a solution with a strictly lower value for objective 1 or 2, another wave of (S + D + 1) improvement attempts is conducted; otherwise the process terminates at a locally optimal solution. Moves to inferior neighboring solutions are forbidden. As mentioned earlier, R locally optimal solutions—one for each of R restarts—are generated for each chromosome in each generation, and the best one determines the chromosome's fitness. After the fitness is computed, the total amount supplied by supplier s in the best of the R locally optimal solutions is stored as the final value of gene s for that chromosome. This is done because a type 2 improvement may change the total amount supplied by supplier s to a value different than the value (G_s) that was used to create the chromosome's initial feasible solutions.

5.2.2. Making the next generation

In the second half of each GA iteration, we first sort the chromosomes in the current generation according to their fitness values. Then the C_1 best chromosomes in the current generation are copied into the next generation. Next, C_2 *extreme chromosomes* are randomly generated and added to the next generation. Each extreme chromosome is such that, with the exception of one s, G_s for all s from 1 to S is either 0 or $(MaxNumS_s)(SQ_s)$. In other words, the total amount supplied by each supplier except one is either zero or the maximum possible amount. Each such chromosome is formed by (i) creating a randomly scrambled list of the S suppliers, (ii) letting $G_s = (MaxNumS_s)(SQ_s)$ for the suppliers at the beginning of the list, (iii) letting $G_s = 0$ for the suppliers at the end of the list, and (iv) identifying the one supplier in the middle of the list whose value G_s satisfies $0 \le G_s \le (MaxNumS_s)(SQ_s)$ such that the sum of all G_s is $MinTotalD$.

Next, parent chromosomes from the current generation are mated, and a total of C_3 ($= C - C_1 - C_2$) children are added to the next generation. In each crossover operation, two unique parent chromosomes ($G1_1$, $G1_2$, $G1_3$, …, $G1_S$) and ($G2_1$, $G2_2$, $G2_3$, …, $G2_S$) are mated to form one child that is a random weighted average of the two parents, with each gene rounded up or down to the next integer to ensure feasibility. A random real number λ in the interval (0,1) decides the weighted average. The "initial child" is ($\lambda G1_1+(1\text{-}\lambda)G2_1$, $\lambda G1_2+(1\text{-}\lambda)G2_2$, $\lambda G1_3+(1\text{-}\lambda)G2_3$, …, $\lambda G1_S+(1\text{-}\lambda)G2_S$); its genes sum to $MinTotalD$ because the genes of each parent sum to $MinTotalD$. Let $SetG$ be the set of genes whose values are non-integer in the initial child. Each gene value in the initial child is then rounded up to the nearest integer. Let $TotalSupplied$ be the sum of the gene values in the resulting "rounded up child." Next, ($TotalSupplied - MinTotalD$) of the suppliers in $SetG$ are randomly selected, and the values of their genes are each reduced by 1. The resulting "final child" is a feasible, "integerized" version of the initial child.

Each parent's selection probability is proportional to its fitness ranking in current generation where the chromosome in the current generation with the best (worst) fitness (ties are broken randomly) has ranking C (1). When a predefined time limit is reached, the GA procedure terminates and the best locally optimal solution that was created is displayed.

It is important to note that the C_2 extreme chromosomes and C_3 child chromosomes in each generation promote the complimentary goals of diversification and intensification within the GA. Children are weighted averages of their parents and are less extreme than their parents; they help to intensify the search of promising areas within the feasible region. Extreme chromosomes, on the other hand, are akin to the "extreme points" or "basic feasible solutions" considered by the simplex algorithm in linear programming. They cannot be formed by taking a weighted average of two or more unique chromosomes; they promote diversity in the GA search so it does not get stuck in a local optimum.

6. Experimental setup, results, and discussion

The experiments consider five methods for solving Math Model #2: (1) pure integer programming (IP) using IBM ILOG CPLEX 12.5; (2) IP where a random feasible solution generated by the method from Sect. 4.2 is provided to the solver at the outset; (3) the GA from Sect. 5.2; (4) the SA algorithm from Sect. 5.1; and (5) the random algorithm. These are referred to as methods 1-5 respectively.

6.1. Generating problem instances

Table 11 shows the parameter values used for generating the problem instances. In all instances, D and S equal 2, 6, 10, 20, or 60, and T equals 10, 30, or 100. In each instance, the demand and supply timing requirements (DT_d and ST_s) and quantities (DQ_d and SQ_s) are randomly generated using the discrete uniform (DU) distribution whose minimum and maximum values are displayed in Table 11. Only feasible instances that satisfy Theorem 1 are considered in the experiments.

Two instance difficulty levels are considered in the experiments. *Easy instances* satisfy no special requirement except being feasible. *Hard instances* must be feasible and satisfy $MaxTotalS - MinTotalD \leq 10$. This requirement limits the decision maker's choices regarding supply subtraction epochs.

A ten-digit code of the form "dDDsSStTTT" is used to indicate the *size* of a problem instance. In this code, "DD" indicates the number of demanders D, "SS" the number of suppliers S, and "TTT" the cycle length T. For example, the code "d02s02t010" denotes a problem with two demanders, two suppliers, and ten periods. Text files defining all instances are provided in the online supplementary material. Parameter (D, S, DQ_d, DT_d, SQ_s, ST_s, T) is shown in row (1, 2, 3, 4, 5, 6, 7) of each file.

Table 11. Parameter values used in the experiments

Parameter	Possible values	Number of possible values
D, S	2, 6, 10, 20, 60	5
T	10, 30, 100	3
DT_d	DU(2, 9)	8
DQ_d	DU(1, 9)	9
ST_s	DU(2, 9)	8
SQ_s	DU(1, 9)	9

6.2. Hardware settings, software settings, termination criteria, SA and GA settings

The experiments are executed in the Windows 7 environment on desktop PCs that have an 8-core, Intel i7-4770, 3.4 GHz CPU with 16 GB of RAM. All solution methods are coded using Microsoft Visual C++ 2010 Professional. IBM ILOG Concert Technology is used within C++ to call IBM ILOG CPLEX 12.5. Default CPLEX settings are used in all cases. All algorithms terminate after 60 seconds have elapsed.

The values of the parameters that guide the SA algorithm and GA were decided based on preliminary experiments whose results are not shown here. Based on these experiments, we use $P = 1000$ and $\alpha = 0.999$ for the initial temperature and cooling factor respectively in the SA algorithm. In the GA,

$(C_1, C_2, C_3) = (5, 15, 30)$ and $R = 10$. That is, the number of (copied, extreme, child) chromosomes in each generation is (5, 15, 30) respectively, and the number of restarts per chromosome is ten.

6.3. Results for easy problem instances

In the first set of experiments, we consider twelve problem sizes corresponding to all possible combinations of four values for *D* and *S*—2, 6, 20, or 60—and three cycle lengths—10, 30, or 100 periods. Ten instances are considered for each problem size. Thus, a total of 120 easy problem instances are considered. Each instance is considered in two ways: using objective 1 and using objective 2. The combination of an instance and objective is called a *scenario*. All five solution methods are tested on all scenarios, yielding 1200 experiments in all.

Table 12 shows the results. The first two columns show the problem size and objective considered. The next five columns show the objective value of the best solution identified by each method for each problem size and objective, averaged across ten instances. The hyphens ("-") in the column of results for method 1—pure CPLEX—indicate that no feasible solution was found within the 60-second time limit for one or more instances of that problem size. The final column shows the number of scenarios (out of ten) in which method 1 identified an optimal solution within the time limit. The final rows show (a) the average objective value achieved by each method for objectives 1 and 2 averaged across all problem sizes; (b) the total number of scenarios (out of 240) in which each method found a solution that was the best among the solutions produced by the five methods (ties included); and (c) the total number of scenarios (out of 240) in which each method failed to identify a feasible solution. Highlighted values indicate the best performing method(s) within a given row of results.

According to Table 12, the overall ranking of the methods from best to worst is GA, CPLEX, SA, and Random. The GA performs the best from at least three vantage points. First, the highlighted cells show that the GA is the best performing method for 18 of the 24 combinations of problem size and objective, whereas method (1, 2, 4, 5) is the best performing method for only (17, 14, 1, 0) of the 24

Table 12. Experimental results for easy instances (all methods terminate after 60 seconds)

Problem Size	Objective	Average Best Objective Value Across Ten Instances					No. Scenarios in which Optimal Solutions Found by Pure CPLEX
		Pure CPLEX	CPLEX w/ Initial Feas. Soln.	GA	SA	Random	
d02s02t010	Objective 1	**3.1**	**3.1**	**3.1**	4.1	3.8	10
	Objective 2	**1.2**	**1.2**	**1.2**	1.7	1.5	10
d02s02t030	Objective 1	**25.2**	**25.2**	**25.2**	33.3	42.2	10
	Objective 2	**2.4**	**2.4**	**2.4**	3.4	4	10
d02s02t100	Objective 1	98.8	98.8	**98.6**	216	281.6	6
	Objective 2	**3.3**	**3.3**	**3.3**	5.5	6.7	10
d06s06t010	Objective 1	**0.1**	**0.1**	**0.1**	2.4	7.2	10
	Objective 2	**0.1**	**0.1**	**0.1**	0.7	2	10
d06s06t030	Objective 1	**3.9**	**3.9**	4.5	36.9	83.9	9
	Objective 2	**0.7**	**0.7**	0.8	3	6.8	9
d06s06t100	Objective 1	**10.9**	12.7	34.4	336.7	591.2	7
	Objective 2	**1.1**	1.3	2.8	7.9	13.8	6
d20s20t010	Objective 1	**0**	**0**	**0**	0.8	18.2	10
	Objective 2	**0**	**0**	**0**	**0**	4.7	10
d20s20t030	Objective 1	4.5	**4.1**	4.7	46.4	183.8	6
	Objective 2	**1**	1.1	1.2	2.9	14.3	5
d20s20t100	Objective 1	206.2	313	**71.5**	382.3	1191.4	4
	Objective 2	-	13.5	**5**	9.4	26.4	4
d60s60t010	Objective 1	**0**	**0**	**0**	1.2	46.4	10
	Objective 2	**0**	**0**	**0**	0.2	10.3	10
d60s60t030	Objective 1	**0.3**	1.7	**0.3**	40.1	345.9	9
	Objective 2	-	5.8	**0.1**	2.7	27.7	6
d60s60t100	Objective 1	-	2673.3	**23.4**	374.2	2039.5	0
	Objective 2	-	73.8	**3**	12.9	45.8	1
Overall Avg Value of Obj 1		-	261.33	**22.15**	122.87	402.93	
Overall Avg Value of Obj 2		-	8.6	**1.66**	4.19	13.67	
No. best solutions found (out of 240 scenarios)		**199**	194	198	62	22	
No. scenarios w/o feasible solution (out of 240)		19	**0**	**0**	**0**	**0**	

Bold: Best performance among the five methods for a given row

combinations. Second, the GA obtains, by far, the best average value for objective 1 (= 22.15) and 2 (= 1.66) across all 120 problem instances. Third, the GA achieves the best balance between number of best solutions identified (= 198) and number of scenarios with no feasible solution identified (= 0) among the five methods. Note that the CPLEX-based methods generally outperform the SA algorithm. Indeed, the CPLEX-based methods vastly outperform SA from the first and third vantage points but perform worse than SA from the second vantage point. As expected, the random algorithm is the worst performing method. It finds optimal solutions in only 21 of 240 scenarios, and the best feasible solutions it finds in the remaining 219 scenarios are of poor quality. It is important to note that, on average, the random algorithm considers 39% more feasible solutions within the 60-second time limit than the SA algorithm for the same problem instance. This indicates that the superiority of SA over the random algorithm is not due to the total number of feasible solutions it considers, but due to its advanced searching ability.

Surprisingly, even if we provide a feasible solution to CPLEX at the outset, there is a chance that it will lead to a worse result than using pure CPLEX. Nevertheless, method 2—CPLEX initialized with a feasible solution—is generally better than method 1—pure CPLEX—because it finds feasible solutions in all scenarios, whereas method 1 fails to identify a feasible solution in 19 scenarios.

Table 13 shows the detailed results when method 1—pure CPLEX—is used to solve each of the 120 easy problem instances. The first two columns show the problem size and objective considered. The next ten columns show the objective value of the best solution identified by method 1 for each instance of that problem size. Provably optimal values are shown in bold.

The many zeros in Table 13 indicate that most of these randomly generated, feasible instances have zero inventory solutions. This is particularly true when *T* is low and *D* and *S* are high. In such cases, there is a high likelihood that the many demand and supply processes can be scheduled so that inventory can be avoided during every period of the short (e.g. 10-period) cycle. These results motivate the creation of more difficult problem instances—considered in Section 6.4—that are less likely to have zero inventory solutions.

Table 13 shows that, as expected, the problems get harder as *T* increases. Indeed, pure CPLEX finds optimal solutions to all instances with *T* = 10, regardless of the values of *D* or *S*, but finds progressively fewer optimal solutions as *T* increases beyond that. Interestingly, the problems do not always get harder as *D* and *S* increase. For example, pure CPLEX's success rate in finding optimal solutions when $D = S = 20$ and $T = 30$ (55%) is less than when $D = S = 60$ and $T = 30$ (75%).

Table 13. Detailed results for solution method 1 (pure CPLEX) on the easy instances

Problem Size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t010	OBJ1	**2**	**0**	**0**	**10**	**0**	**4**	**10**	**0**	**5**	**0**	3.1
	OBJ2	**1**	**0**	**0**	**3**	**0**	**1**	**3**	**0**	**4**	**0**	1.2
d02s02t030	OBJ1	**0**	**0**	**8**	**65**	**90**	**0**	**0**	**0**	**55**	**34**	25.2
	OBJ2	**0**	**0**	**2**	**5**	**6**	**0**	**0**	**0**	**6**	**5**	2.4
d02s02t100	OBJ1	**0**	**0**	**0**	246	**18**	77	**162**	**342**	2	141	98.8
	OBJ2	**0**	**0**	**0**	**7**	**2**	**2**	**8**	**8**	**1**	**5**	3.3
d06s06t010	OBJ1	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.1
	OBJ2	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.1
d06s06t030	OBJ1	**0**	**0**	32	**0**	**0**	**0**	**0**	**7**	**0**	**0**	3.9
	OBJ2	**0**	**0**	5	**0**	**0**	**0**	**0**	**2**	**0**	**0**	0.7
d06s06t100	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	16	81	12	10.9
	OBJ2	**0**	**0**	**0**	**0**	**0**	2	**0**	3	5	1	1.1
d20s20t010	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.0
d20s20t030	OBJ1	11	**0**	**0**	16	8	**0**	**0**	**0**	**0**	10	4.5
	OBJ2	2	**0**	1	2	2	**0**	**0**	**0**	**0**	3	1.0
d20s20t100	OBJ1	152	210	209	**0**	**0**	662	**0**	355	**0**	474	206.2
	OBJ2	N/A	18	25	**0**	**0**	25	**0**	19	**0**	23	-
d60s60t010	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.0
d60s60t030	OBJ1	**0**	**0**	**0**	3	**0**	**0**	**0**	**0**	**0**	**0**	0.3
	OBJ2	**0**	6	N/A	N/A	**0**	**0**	**0**	6	**0**	**0**	-
d60s60t100	OBJ1	N/A	1702	N/A	356	N/A	N/A	346	N/A	N/A	N/A	-
	OBJ2	N/A	**0**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	-

N/A: Can't find any feasible solution within 60 seconds.
Bold: Optimal solution

6.4. Results for hard problem instances

The results from Section 6.3 show that CPLEX performs well on most instances but does not perform well on the largest instances. In addition, most of the randomly generated instances had zero inventory solutions (see Table 13). The purpose of this section is to analyze the performance of the solution methods on more difficult problem instances that do not have zero inventory solutions. Toward this end, we searched for other factors besides problem size that impact problem difficulty. During this search, we found that instances become harder to solve to optimality when *MaxTotalS* is close to *MinTotalD*. Consequently, five new sets of randomly generated, feasible problem instances with $MaxTotalS - MinTotalD \leq 10$ were created. Ten instances are considered for each set. Thus, a total of 50 hard problem instances are considered. The sizes of these instances are different than the sizes of the easy problem instances. Each instance is considered in two ways: using objective 1 and using objective 2. All five solution methods are tested on all instances, yielding 500 experiments in all.

Table 14 shows the results. The structure of this table is identical to Table 12. Note that the success rate in finding optimal solutions is much smaller for these hard instances than the easy instances. For example, the right column shows that CPLEX finds optimal solutions in only 29 of the 100 scenarios.

The relative performance of the solution methods on the hard instances is the same as for the easy instances. In particular, the GA is once again the best performing method from all perspectives. Importantly, the GA defeats the other methods in more convincing fashion for these hard instances than for the easy instances. Once again SA performs better than the random algorithm but generally not as well as the CPLEX-based methods. Indeed, the CPLEX-based methods have a higher success rate (29%) in finding optimal solutions than SA (13%). Also, the average objective value found by CPLEX is usually better than that found by SA. Indeed, the CPLEX-based methods identify better solutions than SA for 9 of the 10 combinations of problem size and objective, whereas SA outperforms CPLEX only for the largest problem size when objective 2 is considered. As expected, the random algorithm is the worst performing method.

Table 15 shows the detailed results when method 1—pure CPLEX—is used to solve each of the 50 hard problem instances. The structure of this table is identical to Table 13. Note that there are proportionally fewer zeros in Table 15 than Table 13. This result agrees with our intuition that requiring $MaxTotalS - MinTotalD \leq 10$ should reduce the likelihood that a zero inventory solution exists to a

Table 14. Experimental results for hard instances (all methods terminate after 60 seconds)

Problem size	Objective	Average Best Objective Value Across Ten Instances					No. Scenarios in which Optimal Solutions Found by Pure CPLEX
		Pure CPLEX	CPLEX w/ Initial Feas. Soln.	GA	SA	Random	
d06s06t030	Objective 1	**30.6**	31.9	32	64.4	93.6	4
	Objective 2	**4.2**	4.4	**4.2**	5.2	8.1	5
d06s06t100	Objective 1	282.4	283.4	**240.7**	335.4	512.5	0
	Objective 2	8.7	8.5	**7.6**	8.8	11.7	0
d10s10t010	Objective 1	**0**	**0**	**0**	2	9.8	10
	Objective 2	**0**	**0**	**0**	0.2	2.7	10
d10s10t030	Objective 1	17.6	**16.6**	21.1	49.2	105.6	0
	Objective 2	3	3.3	**2.6**	3.5	9.2	0
d10s10t100	Objective 1	345.4	342.8	**225.7**	353.3	667.8	0
	Objective 2	11	10.7	**8**	**8**	15.6	0
Overall Avg Value of Obj 1		135.12	134.94	**103.9**	160.86	277.86	
Overall Avg Value of Obj 2		5.38	5.38	**4.48**	5.14	9.46	
No. best solutions found (out of 100 scenarios)		55	48	**82**	25	0	
No. scenarios w/o feasible solution (out of 240)		**0**	**0**	**0**	**0**	**0**	

Bold: Best performance among the five methods for a given row

randomly generated problem instance. The effect of adding this constraint is most directly seen in the results for the problem sizes d06s06t030 and d06s06t100. Whereas CPLEX is able to identify zero inventory solutions in 72.5% of the scenarios for the easy instances of these sizes (Table 13), CPLEX is unable to identify any zero inventory solutions to the hard instances of these sizes (Table 15). In addition, CPLEX has a much lower success rate in identifying optimal solutions to the hard instances of these sizes (9 of 40 scenarios) than the easy instances of these sizes (31 out of 40 scenarios). Furthermore, the best objective values found for the hard instances are much higher than for the easy instances. The above observations indicate that the terms "hard" and "easy" correctly describe the instances in which we do (do not) require $MaxTotalS - MinTotalD \leq 10$. Our final observation in Table 15 is that, as before, the instances with a low T and a high D and S are more likely to have zero inventory solutions.

Table 15. Detailed results for solution method 1 (pure CPLEX) on the hard instances

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d06s06t030	OBJ1	41	25	**45**	17	11	14	26	**48**	**33**	**46**	30.6
	OBJ2	**4**	3	**6**	3	2	2	4	**6**	**6**	**6**	4.2
d06s06t100	OBJ1	377	412	343	243	299	150	275	209	217	303	282.4
	OBJ2	11	9	10	8	9	7	11	7	6	9	8.7
d10s10t010	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
d10s10t030	OBJ1	33	9	10	4	20	2	33	31	33	1	17.6
	OBJ2	4	3	3	2	3	3	4	3	4	1	3
d10s10t100	OBJ1	377	353	372	381	343	313	447	270	239	359	345.4
	OBJ2	10	10	10	11	17	11	13	9	8	11	11

Bold: Optimal solution

The overall results from Sections 6.3 and 6.4 indicate that CPLEX's branch-and-cut algorithm is a good method for attacking problem CICP-MFBSDP-1. However, this method shows limitations on instances that are large (with $S = D \geq 10$) and/or hard (with $MaxTotalS - MinTotalD \leq 10$). In such cases, a good heuristic—such as the GA described in Section 5.2—generally outperforms CPLEX. However, a more basic heuristic—such as the SA algorithm described in Section 5.1—may perform worse than CPLEX across nearly all problem sizes and difficulty levels. Importantly, the GA's superiority over SA appears to be its (1) highly focused local search which is narrower than the SA local search; (2) refusal to accept inferior solutions during the local search; and (3) better separation of overall solution features (e.g. the total amount supplied by supplier s, which is coded in the chromosome) from more detailed solution features (e.g. the supplier/demander timing and batch sizes, which are improved via local search when chromosome fitness is computed).

7. Conclusion

This paper introduces a new deterministic optimization problem to the operations literature: the cyclic inventory control problem with multiple flexible batch supply and demand processes (CICP-MFBSDP). The problem is noteworthy in that the decision maker has control over the timing and lot sizes of all supply and demand processes subject to (1) minimum frequency and batch size requirements for each demand process and (2) maximum frequency and batch size capabilities for each supply process. Thus, demand is *flexible*; it is a control action that the decision maker applies to optimize the system.

The problem is formally defined and modeled as an integer linear program. Two theorems are proved, one which provides a basis for tightening the mathematical formulation. Two versions of the problem are identified: CICP-MFBSDP-1 and CICP-MFBSDP-2. Five solution methods—two exact and three heuristic—are considered for solving problem CICP-MFBSDP-1; these methods are compared in two sets of experiments.

Results show that CPLEX's default branch-and-cut algorithm is a good method for attacking problem CICP-MFBSDP-1. However, this method shows limitations on large instances and instances where the maximum total supply capability *MaxTotalS* is close to the minimum total demand requirement *MinTotalD*. In such cases, a good heuristic—such as the genetic algorithm presented in Section 5.2—generally outperforms CPLEX.

This research could be extended in several ways. First, we could simultaneously consider minimizing the average and maximum inventory levels by applying a nonzero weight to both parts of the objective function ($W_1 > 0$ and $W_2 > 0$). Second, we could incorporate more aspects such as delivery distances and delays into problem CICP-MFBSDP-1. Third, we could develop methods for attacking problem CICP-MFBSDP-2. This relaxed version of the problem permits better coordination between supply and demand—leading to higher throughout with lower inventories—in some cases. Fourth, we could modify the problem so T is a decision variable whose value may be adjusted to better synchronize the supply and demand processes. Fifth, we could investigate other inventory control problems with flexible deterministic demand. Finally, we could consider a situation with *flexible stochastic demand* in which a manager controls not the exact supply and demand timing/amount, but the parameters or other features governing stochastic demand and supply processes—for example, by introducing random interruptions in supply and demand due to the temporary non-availability of a supplier or demander.

Acknowledgements

The authors wish to thank three anonymous referees for several very helpful comments that strengthened this paper.

References

1. Akrami, B., Karimi, B., & Hosseini, S. M. M. (2006). Two metaheuristic methods for the common cycle economic lot sizing and scheduling in flexible flow shops with limited intermediate buffers: The finite horizon case. *Applied Mathematics and Computation, 183*(1), 634-645.
2. Banerjee, A., & Burton, J. (1994). A coordinated order-up-to inventory control policy for a single supplier and multiple buyers using electronic data interchange. *International Journal of Production Economics, 35*, 85-91.
3. Belvaux, G., & Wolsey, L. A. (2000). A specialized branch-and-cut system for lot-sizing problems. *Management Science, 46*(5), 724-738.
4. Ben-Daya, M., Darwish, M., & Ertogral, K. (2008). The joint economic lot sizing problem: Review and extensions. *European Journal of Operational Research, 185*, 726-742.
5. Bitran, G., & Yanasse, H. H. (1982). Computational complexity of the capacitated lot size problem. *Management Science, 28*(10), 1174-1186.
6. Chen, Z., & Sarker, B. (2010). Multi-vendor integrated procurement-production system under shared transportation and just-in-time delivery system. *Journal of the Operational Research Society, 61*, 1654-1666.
7. Dobson, G., & Yano, C. A. (1994). Cyclic scheduling to minimize inventory in a batch flow line. *European Journal of Operational Research, 75*, 441-461.
8. Drexl, A., & Kimms, A. (1997). Lot sizing and scheduling—survey and extensions. *European Journal of Operational Research, 99*, 221-235.

9. Elmaghraby, S., & Bawle, V. (1972). Optimization of batch ordering under deterministic variable demand. *Management Science, 18*(9), 508-517.

10. Eppen, G., & Martin, R. (1987). Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research, 35*(6), 832-848.

11. Florian, M., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1980). Deterministic production planning: Algorithms and complexity. *Management Science, 26*, 669-679.

12. Gaafar, L. (2006). Applying genetic algorithms to dynamic lot sizing with batch ordering. *Computers & Industrial Engineering, 51*(3), 433-444.

13. Glock, G. H. (2012). The joint economic lot size problem: A review. *International Journal of Production Economics, 135*, 671-686.

14. Graves, S. C. (1987). Safety stocks in manufacturing systems. Sloan School of Management, Massachusetts Institute of Technology.

15. Hindi, K. (1995). Algorithms for capacitated, multi-item lot-sizing without set-ups. *Journal of Operations Research Society, 46*, 465-472.

16. Hsieh, W.-H. (2015). Optimal cyclic control of a buffer between two consecutive non-synchronized manufacturing processes. Master's thesis, University of Wisconsin—Milwaukee.

17. Hui, C.-W., & Gupta, A. (2000). A novel MILP formulation for short-term scheduling of multistage multi-product batch plants. *Computers and Chemical Engineering, 24*, 1611-1617.

18. Jamal, A., & Sarker, B. R. (1993). An optimal batch size for a production system operating under a just-in-time delivery system. *International Journal of Production Economics, 32*, 255-260.

19. Karimi, B., Ghomi, S. M. T. F., & Wilson, J. M. (2003). The capacitated lot sizing problem: a review of models and algorithms. *Omega, 31*, 365-378.

20. Katok, E., Lewis, H., & Harrison, T. (1998). Lot sizing in general assembly systems with setup costs, setup times, and multiple constrained resources. *Management Science, 44*(6), 859-877.

21. Khan, L. R., & Sarker, R. A. (2002). An optimal batch size for a JIT manufacturing system. *Computers & Industrial Engineering, 42*, 127-136.

22. Kleindorfer, P., & Newson, E. (1975). A lower bounding structure for lot-size scheduling problems. *Operations Research, 23*(2), 299-311.

23. Lee, H., Park, N., & Park, J. (2009). A high performance finite capacitated MRP process using a computational grid. *International Journal of Production Research, 47*(8), 2109-2123.

24. Li, C.-L., Hsu, N., & Xiao, W.-Q. (2004). Dynamic lot sizing with batch ordering and truckload discounts. *Operations Research, 52*(4), 639-654.

25. Liu, Y., & Karimi, I. (2007). Scheduling multistage, multiproduct batch plants with nonidentical parallel units and unlimited intermediate storage. *Chemical Engineering Science, 62*(6), 1549-1566.

26. Malakooti, B. (2013). Operations and Production Systems with Multiple Objectives. New York: John Wiley & Sons.

27. McDonald, C. M., & Karimi, I. A. (1997). Planning and scheduling of parallel semicontinuous processes 1. Production planning. *Ind. Eng. Chem. Res., 36*, 2691-2700.

28. Méndez, C., Henning, G., & Cerdá, J. (2001). An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Computers & Chemical Engineering, 25*(4-6), 701-711.

29. Ohno, T. (1988). *Toyota Production System: Beyond Large-scale Production.* Productivity Press: Cambridge.

30. Özdamar, L., & Barbarosoglu, G. (1999). Hybrid heuristics for the multi-stage capacitated lot sizing and loading problem. *Journal of Operations Research Society, 50*, 810-825.

31. Özdamar, L., & Birbil, S. (1998). Hybrid heuristics for the capacitated lot sizing and loading problem with setup times and overtime decisions. *European Journal of Operational Research, 110*(3), 525-547.

32. Quadt, D., & Kuhn, H. (2008). Capacitated lot-sizing with extensions: a review. *4OR, 6*(1), 61-83.

33. Sahling, F., Buschkühlb, L., Tempelmeierb, H., & Helbera, S. (2009). Solving a multi-level capacitated lot sizing problem with multi-period setup carry-over via a fix-and-optimize heuristic. *Computers & Operations Research, 36*(9), 2546-2553.

34. Sarker, B. R., & Balan, C. V. (1999). Operations planning for a multi-stage kanban system. *European Journal of Operational Research, 112*, 284-303.

35. Sawik, T. (2009a). Coordinated supply chain scheduling. *International Journal of Production Economics, 120*, 437-451.

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

37. Singh, D. (1990). Using CFM as a competitive edge. *Automation*, *37*, 64-65.

38. Trigeiro, W., Thomas, L., & McClain, J. (1989). Capacitated lot sizing with setup times. *Management Science, 35*(3), 353-366.

39. Webster, F. (1989). A backorder version of the Wagner–Whitin discrete-demand EOQ algorithm by dynamic programming. *Production and Inventory Management Journal, 30*(4), 1-5.