August 2015

Optimal Cyclic Control of a Buffer Between Two Consecutive Non-Synchronized Manufacturing Processes

WenHuan Hsieh
University of Wisconsin-Milwaukee

OPTIMAL CYCLIC CONTROL OF A BUFFER BETWEEN TWO CONSECUTIVE NON-SYNCHRONIZED MANUFACTURING PROCESSES

by

Wen-Huan Hsieh

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Engineering

at

The University of Wisconsin-Milwaukee

August 2015

ABSTRACT

OPTIMAL CYCLIC CONTROL OF A BUFFER BETWEEN TWO CONSECUTIVE NON-SYNCHRONIZED MANUFACTURING PROCESSES

by

Wen-Huan Hsieh

The University of Wisconsin-Milwaukee, 2015
Under the Supervision of Professor Matthew E.H. Petering

This thesis presents methods for efficiently controlling a buffer that is located between two non-synchronized manufacturing processes. Several machines with different cycle times and/or batch sizes perform each manufacturing process. The overall operation cycles every *T* time units. The first objective of the problem is to minimize the average buffer inventory level during one cycle. The second objective is to minimize the maximum inventory level observed at any point during the cycle. This new optimization problem has not been previously considered in the literature. An integer program is developed to model this problem. In addition, two heuristic methods—a simulated annealing algorithm and random algorithm—are devised for addressing this problem. Extensive experiments are conducted to compare the performance of four methods for attacking this problem: pure integer programming using the solver CPLEX; integer programming where CPLEX is initialized with a feasible solution; simulated annealing; and a random algorithm. The advantages and disadvantages of each method are discussed.

Keywords: buffer control; cyclic scheduling; just-in-time; simulated annealing

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank the following people for their assistance. First, I would thank my advisor, Dr. Matthew Petering, for continuous support during my master's degree study and this research, and for his patience and immense knowledge. His supervision helped my research and writing of this thesis.

Besides my advisor, I would like to thank Dr. Jaejin Jang, Dr. Wilkistar Otieno, and Dr. Ichiro Suzuki for serving on my thesis committee, and for their insightful comments and feedback.

I would also like to thank Dr. Ping-Shun Chen, and UWM Sr. Administrative Specialist, Elisabeth Warras for their encouragement and assistance. My sincere thanks also go to Yi-Chen Lin, who supported me to pursue my master's degree all the way from Taiwan to the USA.

Last but not least, I would like to thank my family—my parents and my sister—for supporting me spiritually throughout the writing of this thesis and my life in general.

CHAPTER 1: INTRODUCTION

1.1 Motivation

All industrial systems operate with significant investments in inventory. Inventory is caused by demands and supplies not being synchronized, which is a basic circumstance between those who demand and those who supply. That is, inventory always exists. Demanders typically want goods as soon as possible when they need it. As a result, suppliers are required to have enough merchandise on hand. However, suppliers often do not have as much inventory as they want because inventories are connected to cost and the limited capacities of warehouses.

In a manufacturing environment, there are many ways in which inventory in the system—also known as work-in-process or WIP—and buffer space between machines can be managed. As a result, a material requirement planning (MRP) procedure is usually adopted that generates a production plan which insures that the exact quantity of the right supplies is available at the desired time. However, in some manufacturing systems the process times are not synchronized and/or the batch sizes for two consecutive processes are not the same. For these types of systems, advanced buffer control strategies are needed. This thesis presents one such advanced buffer control strategy.

The particular environment considered in this thesis is as follows. Consider a generic, two-process manufacturing system that produces a single, discrete product. The product undergoes manufacturing process 1 before undergoing process 2. A set of S parallel machines (i.e. *suppliers*) perform manufacturing process 1. A set of D parallel machines (i.e. *demanders*) perform manufacturing process 2. A buffer with infinite capacity is located between the suppliers and demanders. This buffer stores work-in-process. Time is discretized into time periods (e.g. days). The operations are cyclic, repeating every T days (i.e. time periods). The demand associated with each demander d is defined by two parameters—the demand quantity DQ_d and the demand frequency DF_d.

Demander d is satisfied as long as he/she can take one batch of at least DQ_d items from the buffer every DF_d days or more often for all d. The supply associated with each supplier s is defined by two parameters—the supply quantity SQ_s and the supply frequency SF_s. Supplier s is capable of delivering a batch of at most SQ_s items to the system every SF_s days or less often for all s. Assume that supplies come in at the beginning of the day and are followed immediately by demands. The amount left over after the demand is taken is held as inventory for the entire day. The timing and batch sizes for each demander and supplier are decided by the manager of the manufacturing system. The entire system operates on a T day cycle. The goal is to feasibly satisfy the demands with the available supplies (i.e. to keep the buffer inventory at least 0 every day) while minimizing the total and/or maximum inventory held in the buffer over the entire cycle.

1.2 Research objective

The main objective of this study is to develop and test methods and algorithms that seek to minimize the total inventory level within the system described above. These methods will be benchmarked against a less sophisticated method. A secondary objective of this study is to develop a mathematical formulation of the above problem and to obtain theoretical insights that (1) relate to problem feasibility and that (2) strengthen the mathematical formulation.

1.3 Contribution of the thesis

The contributions of this thesis are the following. First, this thesis introduces a new operational problem that has not been previously considered in the literature. Second, we present a mathematical formulation of this new problem. Third, we derive some theoretical results concerning problem feasibility and improving the initial mathematical formulation. Fourth, we present a method for generating feasible solutions for any problem instance that has a feasible region. Finally, we develop four algorithms for solving the problem: (1) traditional integer programming using the solver IBM ILOG CPLEX; (2) integer programming where the solver is given a feasible solution at the outset; (3) a simulated annealing algorithm; and (4) a simple random algorithm. The performance of these four methods is compared across a variety of problem categories and problem sizes. All proposed methods mentioned in this research focus on minimizing the total and/or maximum inventory held during a cycle.

This study is organized as follows. Section 2 reviews the relevant literatures. Section 3 formally describes the problem; presents an example to illustrate the problem; and presents a mathematical formulation of the problem. Section 4 introduces theory that can be used to automatically generate feasible solutions and strengthen the mathematical formulation. Section 5 introduces four methods for solving the problem. Section 6 presents and discusses the results of experiments that compare the performance of these four methods. Section 7 summarizes this research and proposes future extensions of this work. Figure 1-1 shows the flow of this thesis.

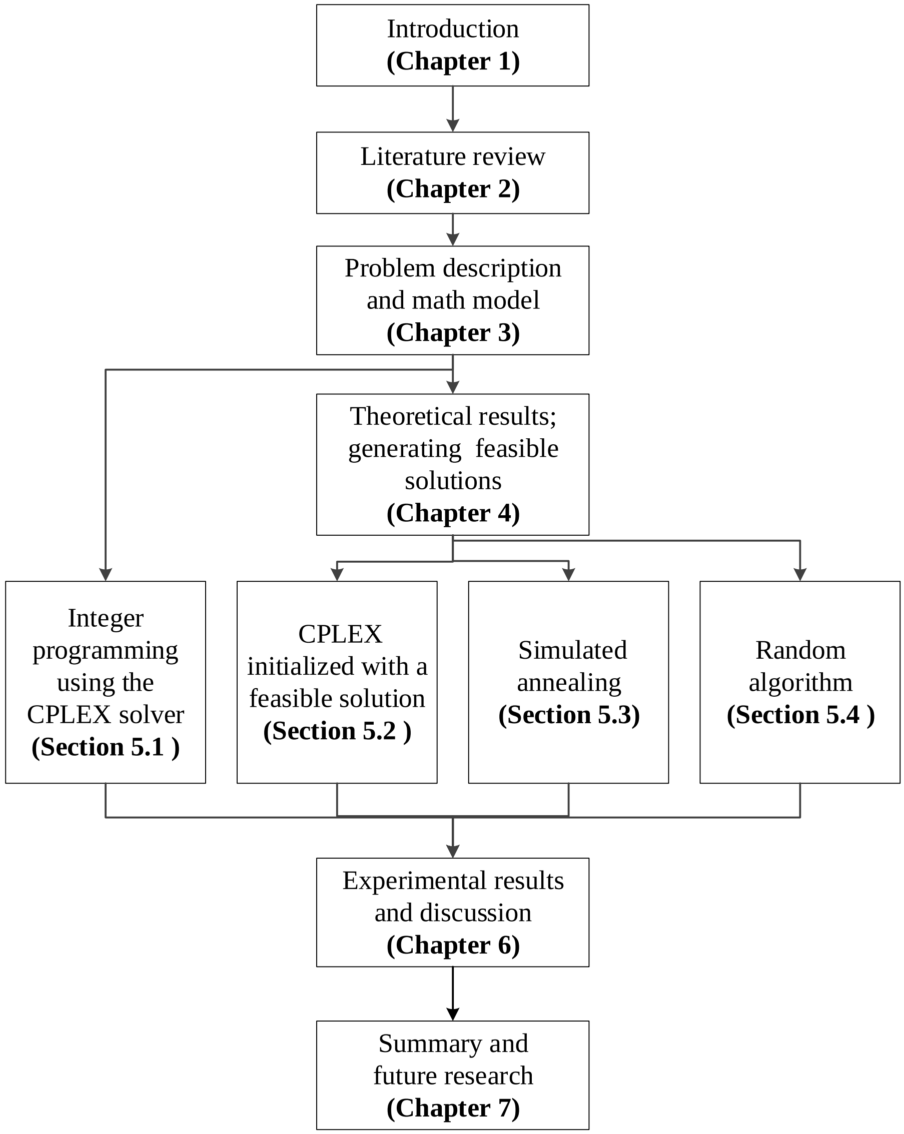

Figure 1-1. Framework of the thesis

CHAPTER 2: LITERATURE REVIEW

The literature relevant to this exploration includes various survey papers on inventory management; papers that consider cyclic inventory systems, just-in-time (JIT) inventory theory, and buffer control; and papers that proposed the original simulated annealing meta-heuristic algorithm for solving various optimization problems.

Major progress in research on supply chain management and inventory management was made at the end of the 20^{th} century when Harris (1990) derived the Economic order Quantity (EOQ) formula that specifies the optimal management protocol for certain types of inventory systems. The EOQ applies when the demand rate is constant. Numerous researchers have elaborated different variations of this EOQ model in recent decades.

Supply and demand inventory optimization problems have been studied broadly under stochastic settings using different methodologies. Florian et al. (1980) consider a class of production planning problems in which known demands have to be satisfied over a finite horizon at minimum total cost. He points out that the problems are NP-hard and unlikely to be solvable in polynomial time. Then he proposes several algorithms and the experimental results are analyzed. Sarker and Parija (1994) consider a manufacturing system which procures raw materials from suppliers and converts them into finished products. The paper develops an ordering policy for raw materials to meet the requirements of a production facility. The objective is to minimize the manufacturing batch size which determines the total cost for making shipments of the finished products.

The organization of the remainder of this chapter is as follows. Section 2.1 provides an overview of cyclic inventory systems. Section 2.2 gives a brief review of the literature on just-in-time (JIT) inventory theory. Section 2.3 reviews the literature on buffer control. Section 2.4 discusses the literature related to simulated annealing.

2.1 Cyclic inventory systems

Graves (1987) describes why cyclic inventory systems are essential: cyclic stock is the inventory in a manufacturing system that exists because production and ordering processes are batch operations. In order to reduce the cyclic stock, the batch size of operations should be reduced. Anticipation stock is the inventory in a manufacturing system intended to smooth the required production rate in the event of a seasonal demand peak exceeding system capacity. To reduce the anticipation stock, the production system must be more closely matched with the cumulative demand placed upon it. Graves elaborates that inventories are the “excess inventories held beyond the minimum inventory level that would be possible in a deterministic and incapacitated world.” As a result, inventory holding is essential because manufacturing systems operate in an uncertain environment.

Whybark and Williams (1976) classify four uncertainties of cyclic inventory systems. The first uncertainty is demand quantity uncertainty. That is, in any given time period, the quantity required of a given part may be different from the planned requirement. Demand quantity uncertainty may result from forecasting errors which require a revision of the master production schedule. The second uncertainty is demand timing uncertainty. This type of uncertainty is present when the expected demand for a given part shifts in time. Demand timing uncertainty may result from changes in the promised delivery date to one or more customers. The third uncertainty is supply quantity uncertainty. This type of uncertainty is present when the quantity of parts available for use is different from the planned quantity. Supply quantity uncertainty may result from unstable yield rates for various in-house manufacturing processes, or from vendors who fail to deliver a promised quantity of raw materials. The last type of uncertainty is supply timing uncertainty. This type of uncertainty is present when the expected set of parts is not available for use exactly when expected. Supply timing uncertainty may result from the variability of in-house production process lead times, or vendors who fail to deliver raw materials on time.

The problem considered in this thesis considers all four of the above uncertainties, but does so from a unique standpoint. Instead of considering these four aspects as random variables, we allow the decision maker to decide these aspects as long as certain requirements can concerning (1) minimum demand quantity, (2) demand timing, (3) maximum supply quantity, and (4) supply timing are met.

Dobson and Yano (1994) consider a cyclic inventory scheduling problem in which there is a constant supply of raw materials and a constant demand for all finished goods. They use a linear programming formulation to determine the optimal cycle length and finishing times for a given set of processes. The objective is to find a production sequence and a cycle length that minimize the average cost per unit time of holding inventory. They assume that inventory can be held at the beginning of the production line, the end of the production line, or between any stations on the line. Xu (2004) provides two approaches to solve a buffer management problem in which demand is uncertain. The first method is make-to-anticipated-order (MTAO), which combines the benefits of the make-to-order (MTO) and anticipated order methods. The second approach is called a postponement and commonality strategy. Mauro (2008) presents a maturity model to develop inventory and operations planning processes for Honeywell Aerospace. This model includes three phases. The first phase is the foundational stage where an initial state with inventory levels based on actual practice is initialized. The second phase, called the right sizing phase, uses traditional single echelon inventory methods to modify the stock levels. Finally, in the third phase, the inventory levels are optimized based on multi-echelon inventory concepts.

2.2 Just-in-time inventory theory

There are four major and common methods to approach inventory (i.e. stock) control: fixed stock level reordering, fixed time re-ordering, economic order quantity, and just-in-time (JIT) inventory control. The philosophy of just-in-time inventory control is to minimize inventory and drive it to zero. That is, the suppliers should only produce exactly the amount required by the demanders. Consequently, the ideal inventory level will be zero and also it can meet the demanders' requirements.

Just-in-time philosophy focuses on the importance minimizing inventory uncertainty, so that the demand quantities and supply quantities match. It is important to realize that the minimizing of demand and supply uncertainty is the goal of JIT, so that inventory safety stocks will no longer be necessary.

Much research has been devoted to evaluating the performance of JIT production systems. Ardalan (1997) and Chu and Shih (1992) use simulations to make evaluations; nevertheless, some researchers have developed analytical methods. Hay (1988) points out that the inventory buffers intended to minimize the impact of production process problems may actually serve to hide these problems from view, and therefore reduce the company's likelihood of taking any steps to solve them. Deleersnyder et al. (1989) analyze a JIT production system using a discrete-time Markov process. Numerical computations are used to study the effects of the number of kanbans, machine reliability, demand variability, and safety stock requirements on the performance of the system. Mitra and Mitrani (1990, 1991) study a multi-stage, serial JIT production system. The subsystem corresponding to each stage is analyzed precisely and an approximation algorithm for finding the best kanban discipline is devised using a decomposition technique.

Wang and Wang (1990) study multi-item JIT production systems using Markovian queues and determine the optimal numbers of kanbans for serial, merge-, or split-type JIT production systems. Halim and Ohta (1994) propose an algorithm to solve batch-scheduling problems to try to minimize inventory cost. In that research, a JIT system is considered and numerical results are presented. Mascolo et al. (1996) use synchronization mechanisms to break down a kanban-controlled production system into a set of subsystems, each of which is analyzed using a product-form approximation. An iterative procedure is developed to determine the performance measures of the overall system. Dong et al. (2001) present an analysis about the impact of JIT theory on supply chain management. The authors introduce a rigorous model to understand under which situations more profit can be achieved using JIT principles. The results show that if suppliers cooperate with each other, they can successfully implement JIT principles to everyone's benefit. Then, they extend the first model via empirical testing. Survey questionnaires are collected and the authors point out that in a JIT system, supply chain integration can improve the buyers' performance, and supplier cooperation can improve the suppliers' performance. Furthermore, if the processes of the suppliers are uncertain and the demand of buyers is certain, or buyers' firms are larger than those of the suppliers, JIT principles have a positive influence. Salameh and Ghattas (2001) mention that the success of the JIT production system lies in the considerable reduction in material inventories that it can achieve. That is, each phase of inventory is highly connected to the total cost, so companies want to minimize the total inventory to reduce the cost of holding inventory. Khan and Sarker (2002) propose an ordering policy for raw materials to meet the requirements of a production facility. First they estimate production batch sizes for a JIT delivery system, and then they incorporate a JIT raw material supply system into the model. A simple algorithm is developed to compute the batch sizes for both manufacturing and raw material purchasing policies.

Chuah (2004) use three heuristic algorithms, including a taboo search algorithm and an ant colony optimization algorithm, to solve a general frequency routing (GFR) problem for a just-in-time supply pickup and delivery system. Matta et al. (2005) consider two different kanban release policies—an independent policy and a simultaneous policy—and compare them by approximate analytical methods. Abuhilal et al. (2006) provide engineering managers with guidelines to choose a cost-effective supply chain inventory control system. They consider push inventory systems (MRP), pull systems (JIT), and MRP with information sharing. Lee et al. (2009) note that executing a production plan at high speed still remains a goal for MRP systems. The authors present the concept of using a computational grid to achieve a breakthrough in MRP performance under conditions of finite capacity. Later, Iwase and Ohno (2011) perform a mathematical evaluation of a multi-stage JIT production system with stochastic demand and limited production capacities. Roy et al. (2012) consider a system where there is a strong bond between a producer and a buyer. An integrated producer-buyer inventory model with constant demand and small lot sizes is considered in two different production environments: an EMQ (economic manufacturing quantity)-based production environment and a JIT-based production environment. The objective is to minimize the inventory level.

Overall, the goal of many JIT-related research papers is to solve inventory problems related to demand and supply imbalance so that inventory levels can be reduced. Having less inventory on hand can reduce cost. The goal of the models and algorithms introduced in this thesis are the same.

2.3 Buffer control

Several papers in the literature investigate buffer control policies within a single facility. Kneppelt (1984) proposes an option overplanting method which requires buffers for storage of, and which increase the safety factor of, sub-assemblies and components in the bill of materials. Newman et al. (1993) argue that companies or factories might be using various “buffers” such as inventory, lead time, and excess capacity to compensate for an inequity between production flexibility and the level of uncertainty in the environment. Buzacott and Shanthikumar (1994) compare using safety stock versus using safety time in a production system and conclude that using safety time is preferable to using safety stock if there is a good prediction of future required shipments. McDonald and Karimi (1997) develop mixed-integer linear programs (MILPs) to minimize the production, inventory, and setup costs for a single facility. Metters (1997) quantifies the bullwhip effect in a supply chain under three inventory control strategies: triggering a new order when there is no inventory; triggering a new order whenever the inventory drops down to the safety stock level; and using a stale safety stock policy. Tang and Grubbström (2002) propose methods for planning and re-planning the master production schedule under stochastic demand to attain a favorable inventory situation. Radhoui et al. (2009) develop a joint quality control and preventive maintenance policy for a randomly failing production system that occasionally produces non-conforming items. Alfieri and Matta (2012) develop mathematical programming formulations that can approximately represent a class of production systems characterized by several stages, limited buffer capacities, and stochastic production times. Fernandez et al. (2013) presents a nonlinear integer programming (NIP) formulation for buffer inventory management to reduce peak electricity consumption without compromising system productivity.

To sum up, hundreds of outstanding articles on inventory control and buffer control can be found in the literature. However, there appears to be no article that studies the same type of system considered in this thesis. In particular, there is no published article that considers the cyclic control of a buffer that lies between two non-synchronized manufacturing processes where a single decision maker can decide the supply frequencies, supply quantities, demand frequencies, and demand quantities as in the present thesis.

2.4 Simulated annealing algorithms

Simulated annealing is a generic probabilistic methodology for finding the global optimum to a large (typically combinational) optimization problem characterized by (1) a huge number of variables; (2) a relatively unconstrained feasible region (where feasible neighboring solutions can be easily generated), and (3) a complex objective function. The method of simulated annealing (SA) was pioneered by Metropolis et al. (1953). The name SA comes from annealing in metallurgy which utilizes heating followed by controlled cooling in order to increase the size of the crystals (and thereby reduce defects) in various metal parts used in industry. This method, however, did not receive much attention at the time. After that, Kirkpatrick et al. (1983) applied these ideas and proposed what we know today as the simulated annealing algorithm.

One feature of SA is that it probabilistically replaces a current solution with a worse neighboring solution so that the search can jump out of a local optimal solution. Consider an optimization problem where the goal is to minimize the objective value. Let the objective value of the current solution be E. Then a perturbation mechanism is applied to create a candidate (i.e. neighboring) solution that is slightly different than the current solution. The candidate objective value E' comes from the neighboring solution. If the difference between these two corresponding values of the objective values, ΔE $(= E' - E)$, is less or equal than zero, then the search is continued with the neighboring solution. Otherwise, if ΔE is greater than zero, the inferior neighboring solution is accepted with probability $exp(-\frac{\Delta E}{P})$ (Kirkpatrick et al., 1983). The parameter P represents the current temperature, which controls the annealing process and the acceptance probability. The temperature is gradually cooled as the procedure unfolds. When the temperature is high, it is easier to accept an inferior neighboring solution; this brings the feature of diversification. When the temperature is low, there is a lower probability of accepting an inferior neighboring solution and the search for a final optimal solution intensifies; this feature is known as intensification.

Overall, simulated annealing has been shown to be an effective method for attacking large optimization problems because it combines the features of diversification and intensification. Simulated annealing is one of the four methods we use to solve the optimization problem introduced in this thesis.

CHAPTER 3: PROBLEM DESCRIPTION AND MATH MODEL

3.1 Problem description

We now formally describe the problem under investigation in this thesis. Consider a generic, two-process manufacturing system that produces a single, discrete product. The product undergoes manufacturing process 1 before undergoing process 2. A set of S parallel machines (i.e. upstream machines, *suppliers*) perform manufacturing process 1. A set of D parallel machines (i.e. downstream machines, *demanders*) perform manufacturing process 2. A *buffer* with infinite capacity is located between the S suppliers (i.e. upstream machines) and D demanders (i.e. downstream machines). This buffer stores work-in-process, i.e. parts that have completed manufacturing process 1 and are waiting to start manufacturing process 2. Time is discretized into time periods (e.g. days). The operations are cyclic, repeating every T days (i.e. time periods). The demand associated with each demander d is defined by two parameters—the *demand quantity* DQ_d and the *demand frequency* DF_d. Demander d is satisfied as long as he/she can take one batch of at least DQ_d items from the buffer every DF_d days or more often for all d. The supply associated with each supplier s is defined by two parameters—the *supply quantity* SQ_s and the *supply frequency* SF_s. Supplier s is capable of delivering a batch of at most SQ_s items to the system every SF_s days or less often for all s. Assume that supplies come in at the beginning of the day and are followed immediately by demands. The amount left over after the demand is taken is held as inventory for the entire day. The demand timing and batch sizes for each demander are decided by the manager of the manufacturing system. The supply timing and batch sizes for each supplier are also decided by the manager of the system. The entire system operates on a T day cycle. The goal is to feasibly satisfy the demands with the available supplies (i.e. to keep the buffer inventory at least 0 every day) while minimizing the total and/or maximum inventory held in the buffer during each cycle.

Figure 3.1 depicts this cyclic system. The buffer is indicated by a solid rectangle in the middle of the diagram. The *S* suppliers comprising manufacturing process 1—each with a unique supply quantity SQ_s and supply frequency SF_s—are shown inside the dotted rectangle on the left. The *D* demanders comprising manufacturing process 2—each with a unique demand quantity DQ_d and demand frequency DF_d—are shown inside the dotted rectangle on the right. As mentioned at the top of the diagram, the overall operation cycles every *T* days. The goal is to minimize the average inventory held in the buffer during a cycle and/or the maximum inventory level achieved at any time during the cycle.

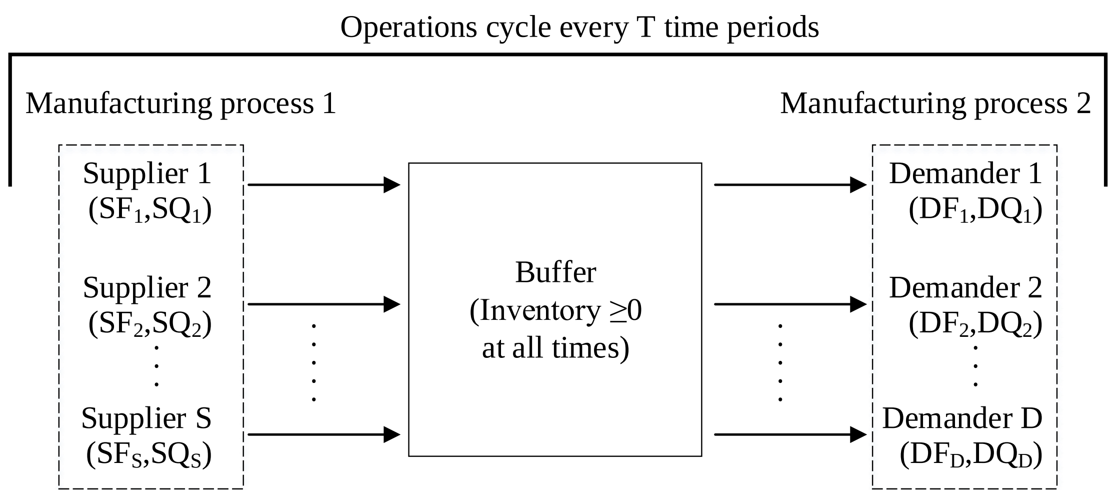

Figure 3-1. System under investigation

3.2 Illustrative example

Tables 3-1 and 3-2 provide an illustration of the problem at hand. The input data for this example are shown in Table 3-1. In this simple problem we have three demanders and three suppliers. The demand frequencies for demanders 1, 2, 3 are 3, 2, 6 days respectively. The demand quantities for demanders 1, 2, 3 are 2, 4, 2 units respectively. The supply frequencies for suppliers 1, 2, 3 are 2, 5, 3 days respectively. The supply quantities for suppliers 1, 2, 3 are 4, 3, 5 units respectively. The required cycle length is 10 days.

Table 3-2 shows a feasible solution for this problem instance. In this solution, demander 1 takes 2 items from the buffer on each of the days T3, T5, T7, and T10; demander 2 takes 4 items on each of days T2, T4, T6, T8 and T10; and demander 3 takes 2 items on each of the days T4 and T10. Note that the batch sizes taken by the demanders—2, 4, and 2 items respectively—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 DF_1, DF_2, and DF_3—3, 2, and 6 days respectively—for demanders 1, 2, and 3 respectively. In Table 3-2, supplier 1 replenishes the buffer with 4, 4, 4, 1, and 3 items at the beginning of days T1, T3, T5, T7, and T9; supplier 2 replenishes the buffer with 3 and 3 items at the beginning of days T2 and T7; and supplier 3 replenishes the buffer with 4, 3, and 3 items at the beginning of days T1, T4, and T8. Note that the amount delivered by supplier 1, 2, and 3 never exceeds the values of SQ_1, SQ_2, and SQ_3—4, 3, and 5 respectively—for suppliers 1, 2, and 3 respectively. Also, the time that elapses between consecutive supply occurrences is never less (i.e. never more often) than the values of SF_1, SF_2, SF_3—2, 5, and 3 days respectively—for suppliers 1, 2, and 3 respectively. Note that the operations cycle every 10 days so that day T1 immediately follows day T10. The inventory held in the buffer during each day is shown in the long row near the bottom of the table. 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 zeroes in Table 3-2 mean that no demand is made or nothing is supplied at that time. The goal is to minimize objective 1 and/or objective 2. The displayed solution is not optimal and is only one of thousands of feasible solutions to this problem instance. One can imagine that this type of problem becomes more difficult to solve to optimality as the number of demanders and suppliers, and the cycle length, increase. Thus, the goal of this thesis is to find a way to solve this challenging problem with an efficient method.

Table 3-1. Illustrative instance #1

# of demanders: 3		#of suppliers: 3	
DQ_1: 2	DF_1: 3	SQ_1: 4	SF_1: 2
DQ_2: 4	DF_2: 2	SQ_2: 3	SF_2: 5
DQ_3: 2	DF_3: 6	SQ_3: 5	SF_3: 3
T=10			

Table 3-2. Feasible solution for illustrative instance #1

	Time Period (Day)									
	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 (Cumulative inventory)	61	Objective 2 (Maximum inventory)	9							

There are seven major elements that define the feasible solution shown in Table 3-2. First, a *demand start point* is the time period in which a demander first initiates a demand. For example, demander 1's start point is T3. Second, a *supply start point* is time period in which a supplier first initiates a supply. For instance, supplier 2’s start point is time period T2. Third, *demand intervals* indicate the time that elapses between demand occurrences beginning with the demand start point. For example, demander 1’s intervals are (2, 2, 3, 3) corresponding to the time between the demand occurrences corresponding to this demander—T3, T5, T7, and T10. Indeed, T3 and T5 are separated by 2 time intervals; T5 and T7 are separated by 2 time intervals; T7 and T10 are separated by 3 time intervals; and T10 and T3 are separated by 3 time intervals. Fourth, *supply intervals* indicate the time that elapses between supply occurrences beginning with the supply start point. For example, supplier 2's intervals are (5, 5). Note that each demander's intervals and each supplier's intervals should sum to T. Also, no demand interval for demander d should exceed DF_d. Also, no supply interval for supplier s should be less than SF_s. Fifth, a *supply subtraction epoch* indicates where the amount actually supplied is less than a supplier's ability to supply. For example, supplier 1 has two subtraction epochs—T7 and T9—where less than the maximum value of SQ_1 (=4) is supplied. Supplier 2, on the other hand, has no subtraction epochs. Finally, the *number of demand (supply) occurrences* is the number of times during the cycle when a demander (supplier) takes a batch of sufficient size from (supplies a batch to) the buffer. For example, the number of demand occurrences for demander 1 is 4, and the number of supply occurrences for supplier 3 is 3.

3.3 Math model

The above situation can be modeled as an integer linear program (ILP). The notations used in this ILP are given in Table 3-3, Table 3-4, and Table 3-5. Table 3-3 displays the indices used in the math model. Index d denotes a demander; index s denotes a supplier; indices t and u denote a time interval; and index e denotes an objective function component. Table 3-4 shows the primary parameters used in the math model: the total number of demanders D; total number of suppliers S; cycle length for the inventory system T; minimum quantity demand per batch for demander d (DQ_d); demand frequency for demander d (DF_d); maximum quantity supplied per batch for supplier s (SQ_s); supply frequency for supplier s (SF_s); and weight for objective function component e (W_e). For example, when W_1 equals 1 and W_2 equals zero, it means that the sole objective is to minimize the total inventory level. Table 3-5 displays the decision variables in the math model. $SYN_{s,t}$ is a binary variable that indicates if supplier s supplies a batch at the beginning of time interval t or not. $SAmt_{s,t}$ is an integer variable that decides the amount supplied by supplier s at the beginning of time interval t. $DYN_{d,t}$ is a binary variable that indicates if demander d demands a batch of sufficient size at the beginning of time interval t or not. $DAmt_{d,t}$ is an integer variable that decides the amount demanded by demander d at the beginning of time interval t. I_t is the inventory on hand during time interval t. *IMax* is the maximum interval level observed during the entire cycle.

Table 3-3. Indices in Math Model #1

d	demander	(d = 1 to D)
s	supplier	(s = 1 to S)
t, u	time interval	(t, u = 1 to T)
e	index of the objective function	(e = 1, 2)

Table 3-4. Parameters in Math Model #1

PRIMARY PARAMETERS	
T	Cycle length for the inventory system (integer, > 0).
D	Number of demanders (integer, >0).
S	Number of suppliers (integer, >0).
DQ_d	Minimum quantity demand per batch for demander d (integer, >0).
DF_d	Demand frequency for demander d (integer, >0, ≤T). Maximum number of days between consecutive batches of sufficient size taken by demander d.
SQ_s	Maximum quantity supplied per batch for supplier s (integer, > 0).
SF_s	Supply frequency for supplier s (integer, > 0, ≤T). Minimum number of days between consecutive batches supplied by supplier s.
W_e	Weight for index e of the objective function (real, ≥0)
SECONDARY PARAMETERS (Derived parameters)	
TotalD	Minimum total quantity that is demanded during the cycle (integer, >0).
TotalS	Maximum total quantity that can be supplied during the cycle (integer, >0).

Table 3-5. Decision variables in Math Model #1

$SYN_{s,t}$	= 1 if supplier *s* supplies a batch at the beginning of time interval *t* (binary).
$SAmt_{s,t}$	Amount supplied by supplier *s* at the beginning of time interval *t* (integer, ≥ 0).
$DYN_{d,t}$	= 1 if demander *d* takes a batch of sufficient size at the beginning of day *t* (binary).
$DAmt_{d,t}$	Amount demanded by demander *d* at the beginning of time interval *t* (integer, ≥ 0).
I_t	Inventory on hand during time interval *t* (integer, ≥ 0).
IMax	Maximum inventory during the cycle (integer, ≥ 0).

Math Model #1:

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

Subject to: $$I_t \le IMax \qquad \forall t \tag{2}$$

$$SAmt_{s,t} \le (SQ_s)(SYN_{s,t}) \qquad \forall s\,\forall t \tag{3}$$

$$DAmt_{d,t} \ge (DQ_d)(DYN_{d,t}) \qquad \forall d\,\forall t \tag{4}$$

$$\sum_{u=t}^{t+SF_s-1} SYN_{s,((u-1)\bmod T)+1} \le 1 \qquad \forall s\,\forall t \tag{5}$$

$$\sum_{u=t}^{t+DF_d-1} DYN_{d,((u-1)\bmod T)+1} \ge 1 \qquad \forall d\,\forall t \tag{6}$$

$$I_1 = I_T + \sum_{s=1}^{S} SAmt_{s,1} - \sum_{d=1}^{D} DAmt_{d,1} \tag{7}$$

$$I_t = I_{t-1} + \sum_{s=1}^{S} SAmt_{s,t} - \sum_{d=1}^{D} DAmt_{d,t} \qquad \forall t : 2 \le t \le T \tag{8}$$

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

3.4 Math model explanation

In Math Model #1, the first objective is to minimize the total item-days of inventory held over the entire cycle of T days, and the second objective is to minimize the maximum inventory level achieved at any time during the cycle (1). Constraint (2) confirms that each inventory level will not exceed the maximum inventory level. Constraint (3) ensures that the amount supplied by supplier s cannot exceed SQ_s on any given day and that supplier s cannot supply anything at the beginning of day t if the variable $SYN_{s,t} = 0$. Constraint (4) ensures that the amount demanded by demander d is at least DQ_d when $DYN_{d,t} = 1$ and is at least 0 when $DYN_{d,t} = 0$. Constraint (5) ensures that at most one batch is supplied by supplier s during any SF_s-day period. Constraint (6) ensures that at least one batch of sufficient size is taken by demander d during any DF_d-day period. Constraints (7-8) ensure that the inventory on hand during each time interval is properly computed. Constraint (9) requires that no inventory be on hand during the final time interval. This constraint eliminates symmetries and redundant solutions that are cycles of each other.

CHAPTER 4: NECESSARY AND SUFFICIENT CONDITIONS FOR PROBLEM FEASIBILITY

4.1 Computation of secondary parameters

The secondary parameters *TotalD* and *TotalS* from Table 3-4 are computed as follows.

$$TotalD = \sum_{d=1}^{D} (NumD_d)(DQ_d) \qquad \text{where} \qquad NumD_d = \left\lceil \frac{T}{DF_d} \right\rceil \quad \forall d. \tag{10}$$

$$TotalS = \sum_{s=1}^{S} (NumS_s)(SQ_s) \qquad \text{where} \qquad NumS_s = \left\lfloor \frac{T}{SF_s} \right\rfloor \quad \forall s. \tag{11}$$

As stated in Table 3-4, *TotalD* is minimum total quantity that is demanded during the cycle (integer, >0). Also, *TotalS* is maximum total quantity that can be supplied during the cycle (integer, >0). $NumD_d$ is minimum number of demand occurrences for demander d during the cycle. It equals the smallest integer greater than or equal to T divided by DF_d. Also, $NumS_s$ is the maximum number of replenishments (i.e. supply occurrences) made by supplier S during the cycle. It equals the largest integer less than or equal to T divided by SF_s.

4.2 Necessary and sufficient conditions for problem feasibility

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

Theorem 4-1: The problem is feasible if and only if TotalS $\geq$ *TotalD.*

Proof: If we sum up constraint (7) and all constraints of type (8) in Math Model #1, we arrive at the following:

$$\sum_{s=1}^{S}\sum_{t=1}^{T} SAmt_{s,t} = \sum_{d=1}^{D}\sum_{t=1}^{T} DAmt_{d,t} \tag{12}$$

In other words, the total amount supplied during the entire cycle should equal the total amount demanded. If *TotalS* < *TotalD*, the above requirement cannot met and the problem is infeasible.

Next, we observe that whenever the maximum total supply quantity *TotalS* is equal to or greater than the minimum total demand quantity *TotalD*, we can always construct a feasible solution. Section 4.3 will present a method to generate such a solution.■

4.3 Method for automatically constructing a feasible solution

In this section, we present a method to automatically generate a random feasible solution to Math Model #1 wherever *TotalS* ≥ *TotalD*. This method is summarized in Table 4-1.

We use the problem instance shown in Table 4-2 to illustrate this method. Assume that there are six demanders and six suppliers and their requirements/capabilities are shown in Table 4-2.

Table 4-1. Summary of procedure for automatically generating a feasible solution

Step	Explanation
1	Generate random demand occurrences and supply occurrences
2	Reduce supplies
3	Build inventory diagram
4	Move X-axis equal to the lowest inventory level
5	Move Y-axis so the final inventory value is 0

Table 4-2. Illustrative instance #2

# of demanders: 6		#of suppliers: 6	
DF_1: 3 ($NumD_1$=6)	DQ_1: 2	SF_1: 9 ($NumS_1$=1)	SQ_1: 7
DF_2: 7 ($NumD_2$=3)	DQ_2: 4	SF_2: 8 ($NumS_2$=2)	SQ_2: 6
DF_3: 6 ($NumD_3$=3)	DQ_3: 2	SF_3: 6 ($NumS_3$=2)	SQ_3: 9
DF_4: 5 ($NumD_4$=4)	DQ_4: 2	SF_4: 7 ($NumS_4$=2)	SQ_4: 4
DF_5: 3 ($NumD_5$=6)	DQ_5: 1	SF_5: 5 ($NumS_5$=3)	SQ_5: 9
DF_6: 4 ($NumD_6$=5)	DQ_6: 7	SF_6: 7 ($NumS_6$=2)	SQ_6: 4
T=17			

In step 1, we generate random demand occurrences and supply occurrences that satisfy constraints (3-6) in Math Model #1. All supplies and demands need to be guided by each quantity and frequency. To satisfy constraints (4) and (6), a random demand start point between 1 and T is selected for each demander. Then, random demand intervals are generated for each demander so as to agree with constraint (6). In particular, for each d, we let demander d's demand intervals be a set of $NumD_d$ random positive integers that sum to T, each of which is $\leq DF_d$. The amount demanded by demander d for each of his/her demand occurrences is set equal to DQ_d for all d. To satisfy constraints (3) and (5), a random supply start point between 1 and T is selected for each supplier. Then, random supply intervals are generated for each supplier so as to agree with constraint (5). In particular, for each s, we let supplier s's supply intervals be a set of $NumS_s$ random positive integers that sum to T, each of which is $\geq SF_s$. The amount supplied by supplier s for each of his/her supply occurrences is set equal to SQ_s for all s. Overall, we randomly arrange each demander's demand intervals and supplier's supply intervals within T cycle days and make sure that the intervals do not violate the demand and supply frequencies specified by DF_d and SF_s. Figure 4-1 shows the result of the above process applied to Illustrative Instance #2. We call this item an *initial supply and demand table*. Note that the total amount supplied in Figure 4-1 is 80 units per cycle and the total amount demanded in the Figure 4-1 is 79 units per cycle. That is, condition (12) is not satisfied.

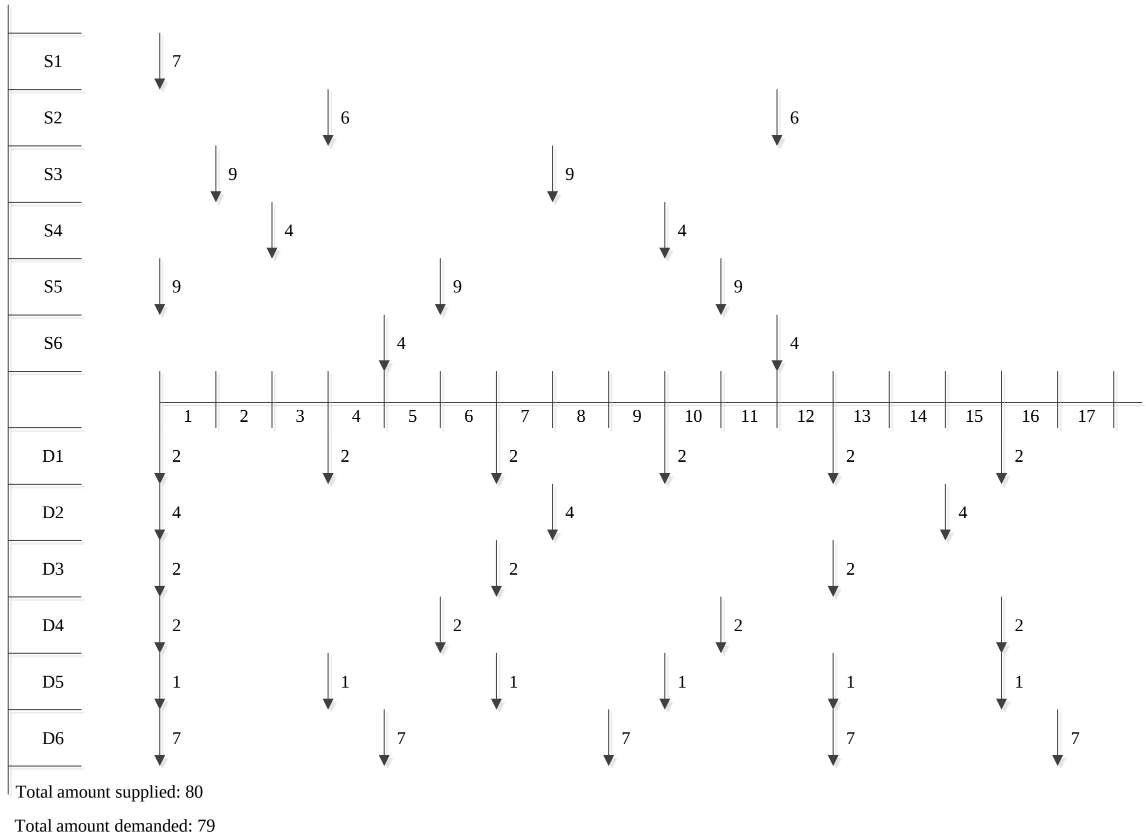

Figure 4-1. Step 1 in procedure for constructing a feasible solution: generate random demands and supplies

In step 2, we reduce some of the supply amounts until the total amount supplied equals the total amount demanded. In other words, if the total supply quantity value is greater than total demand quantities, then we subtract some surplus from some supply occurrences to meet the total demand quantities. In this step we keep randomly deleting a random unit of supply until the total amount supplied during the cycle equals the total amount demanded during the cycle. After this, we obtain a *balanced supply and demand table*, as shown in Figure 4-2. In this table, condition (12) is met. We then sum up the total amount supplied in each time interval and the total amount demanded in each time interval (bottom of Figure 4-2).

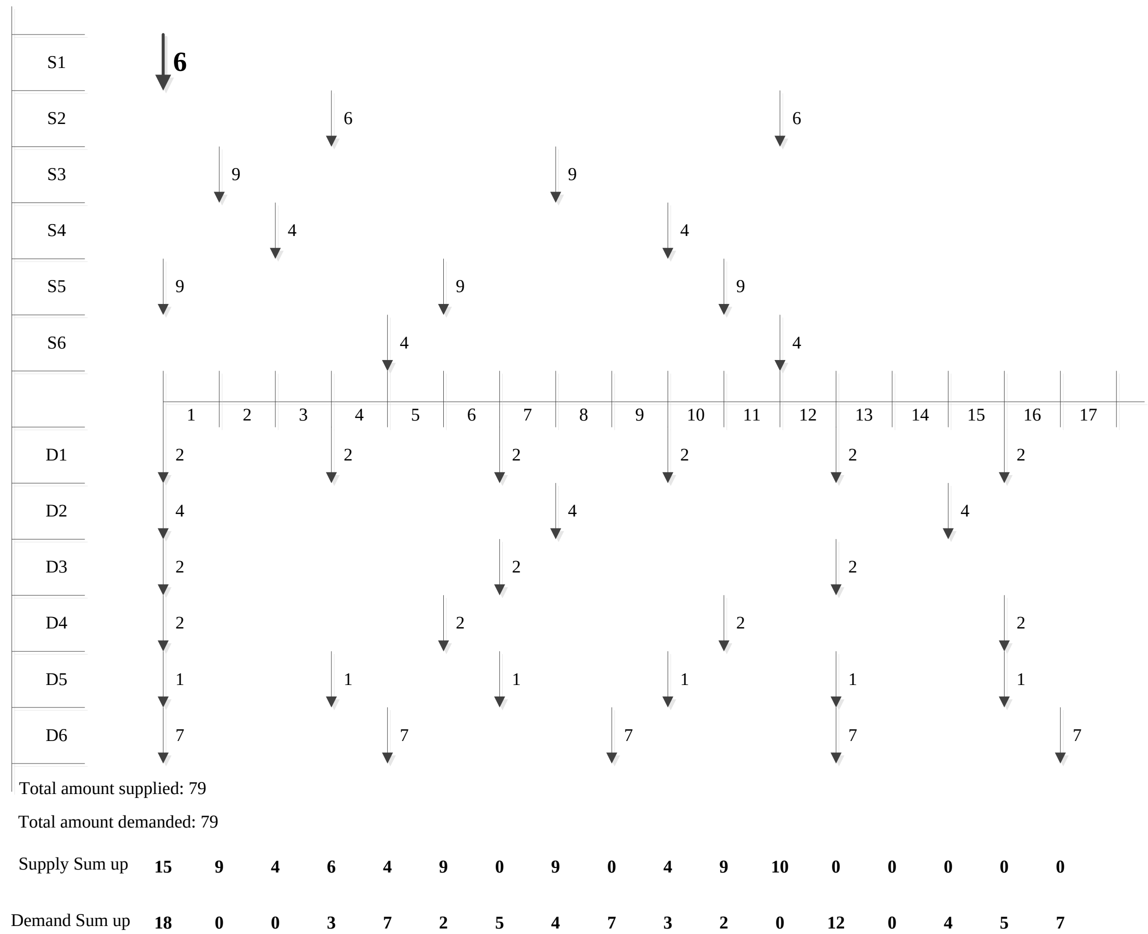

Figure 4-2. Step 2 in procedure for constructing a feasible solution: reduce supplies

In step 3, we first compute the *net amount supplied* (amount supplied minus amount demanded) *during each time period*. This is displayed in the “Balance” row in Figure 4-3. Then, we use these values to compute the inventory on hand during each time interval in the cycle. This is displayed in the “Inventory” row in Figure 4-3. Then, we draw *an initial inventory diagram* that shows the inventory level over the entire cycle. The diagram helps us check for errors or mistakes. Figure 4-3 shows the results.

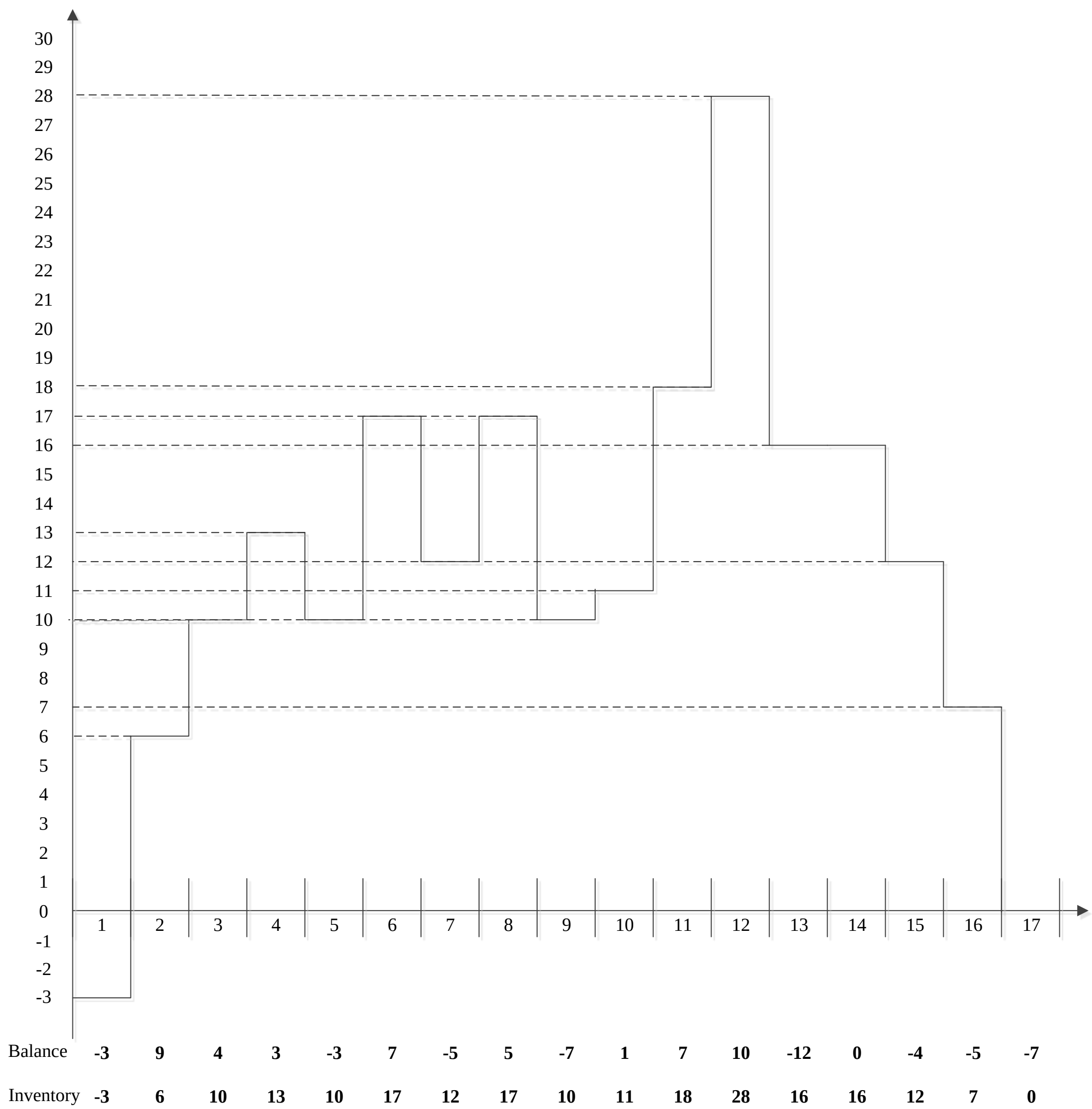

Figure 4-3. Step 3 in procedure for constructing a feasible solution: build inventory diagram

In step 4, we compute the lowest inventory value. Then we subtract the lowest inventory value observed during the cycle from every inventory value in the cycle. A new inventory diagram is then created (see Figure 4-4).

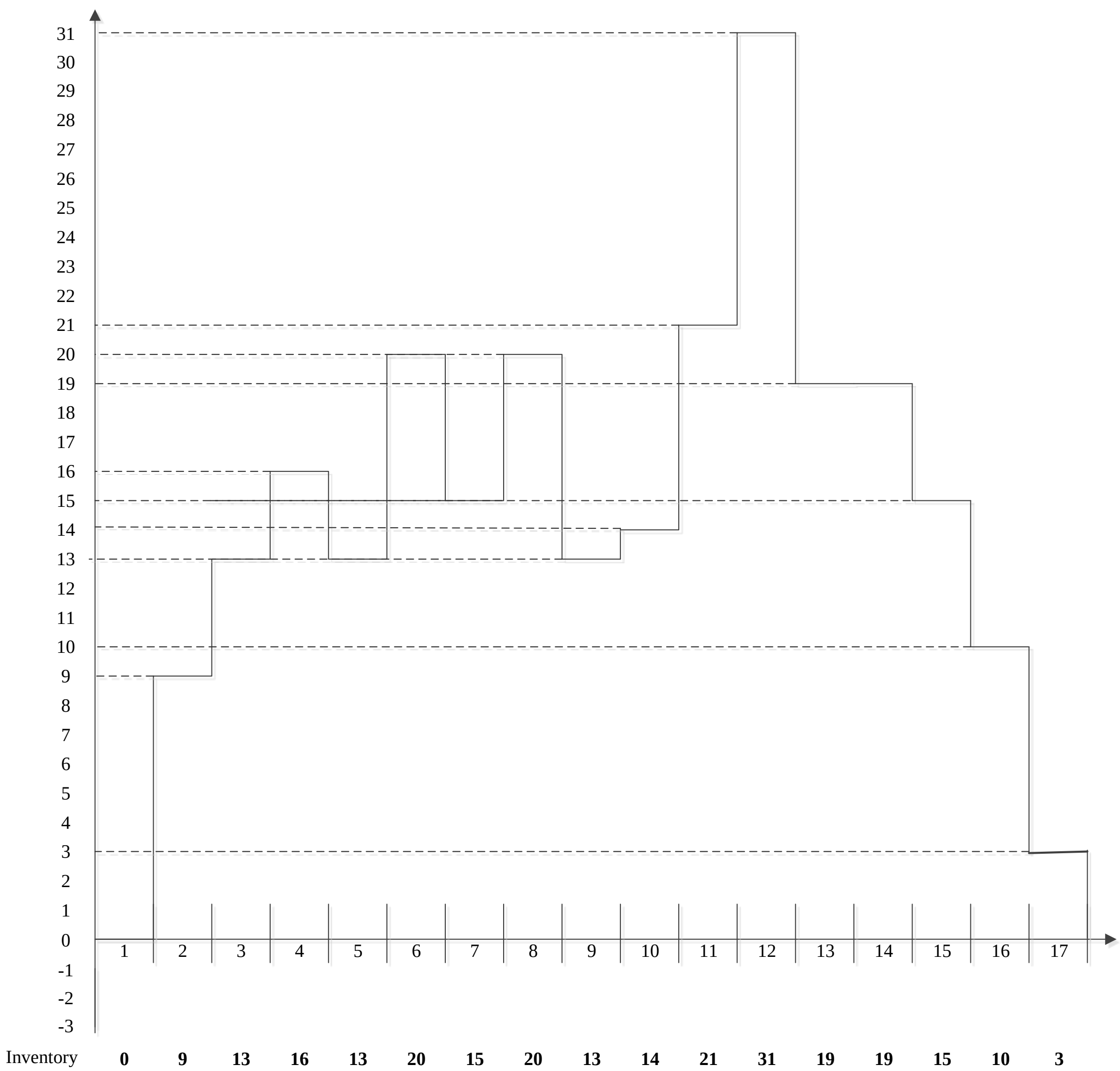

Figure 4-4. Step 4 in procedure for constructing a feasible solution: move x-axis

In step 5, according to constraint (9) in Math Model #1, the last time period of *T* should not have any inventory on hand. Consequently, we use the feature of cyclic systems so that we can move the entire diagram horizon around until the zero phase occurs during the last time period (*T*) as shown in Figure 4-5. Figure 4-5 is the *final inventory diagram* corresponding to the balanced supply and demand table shown in Figure 4-2. It satisfies all constraints in Math Model #1. Then we look at the inventory levels and compute the two objective values. In this example, objective 1’s value is 251 units and objective 2’s value is 31 units.

The above procedure is utilized within three of the four solution methods presented in the next chapter.

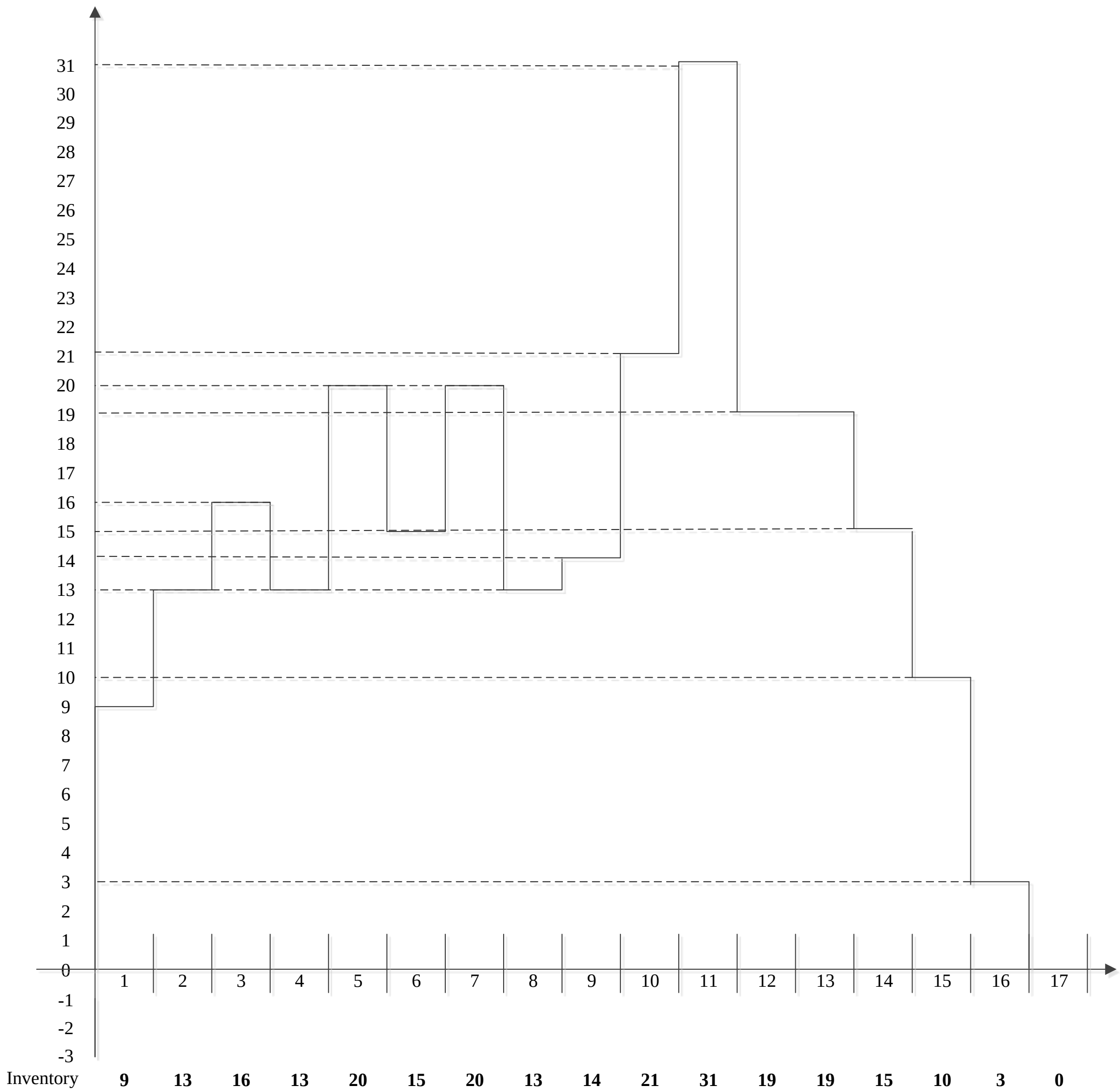

Figure 4-5. Step 5 in procedure for constructing a feasible solution: move y-axis

4.4 Tightening the mathematical formulation

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

Theorem 4-2: 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 demands are just barely satisfied (i.e. the demands are exactly met, i.e. the demand quantities are 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), extra units are demanded but these "extra demands" are not helping to satisfy any constraints in Math Model #1 beyond what the values (i) 0 and (ii) DQ_d would 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 and all t such that the value of objective 1 for Z' is at least as good as that for Z and the value of objective 2 for Z' is at least as good as that for Z. Here is how. In solution Z, consider each "extra unit of demand" one at a time. Delete each such "extra unit of demand", and delete one unit of supply occurring during the same time interval (if possible) or the time interval that is earlier than and as close as possible to this time interval. The resulting solution Z is still feasible and has objectives 1 and 2 at least as good as before.

Table 4-3 shows an example of this process. The top half of the table shows a feasible solution Z for illustrative instance #1 with no "extra units of demand." The value of objective 1 (2) for this solution is (). The bottom half of the table shows feasible solution Z'. for this instance that is obtained using the above process. The values in bold have been changed. Note that the value of objective 1 (2) for solution Z' is (), which is at least as good as the respective value for Z.■

Table 4-3. Example for supporting the proof of Theorem 4-2

(**Z'**)	Time Period (Day)									
	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 (Cumulative inventory)	**61**	Objective 2 (Maximum inventory)		9						
(**Z**)	Time Period (Day)									
	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	**5**	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	5	0	4	3	3	0
SI-DI	8	-1	2	-3	3	-5	2	-1	3	-8
Inventory held	8	7	9	6	9	4	6	5	8	0
Objective 1 (Cumulative inventory)	**62**	Objective 2 (Maximum inventory)		9						

Theorem 4-2 allows us to simplify and tighten Math Model #1 to remove portions of the feasible region that do not include solution Z'. In particular, we can compute the total amount that is demanded per cycle (equal to the total amount supplied per cycle) at the outset prior to solving the problem. Let *TotalD* denote this quantity. Then the following math model can be used instead of Math Model #1 as a correct formulation of this problem.

Math Model #2:

(1), (2), (3), (5), (6), (7), (8), (9) from Math Model #1

$$DAmt_{d,t} = (DQ_d)(DYN_{d,t}) \qquad \forall d \ \forall t \tag{4'}$$

$$\sum_{d=1}^{D}\sum_{t=1}^{T} DAmt_{d,t} = TotalD \tag{13}$$

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

In Math Model #2, constraint (4') specifies that the amount demanded should equal 0 or DQ_d in all cases. Also, constraints (13-14) ensure that both the total amount supplied in the cycle and the total amount demanded in the cycle equal the parameter *TotalD* (Table 3-4 and equation (10)) and no more. From this point onwards, all discussion of math models concerns Math Model #2. Thus, Math Model #2 is the basis of the math-programming-related methods and experiments described in Sections 5.1, 5.2, and 6.

CHAPTER 5: FOUR SOLUTION METHODS

This section describes the various procedures used in the computational study. Overall, a total of four algorithms (i.e. methods) were developed to solve Math Model #2. The first method is pure integer programming using the solver CPLEX. The second method is CPLEX initialized with a feasible solution. The third method is simulated annealing (SA). The fourth method is a random algorithm that provides a benchmark for the SA method.

The procedure from Section 4.3 (which automatically generates a random feasible solution) assists three of the above solution methods. First, it provides the initial feasible solution for the integer programming solver CPLEX. Second, it is embedded within the simulated annealing method. Finally, it is the core of the random algorithm.

5.1 Integer programming using CPLEX

IBM ILOG CPLEX 12.5 is an advanced integer linear programming (ILP) solver that has the ability to efficiently solve problems with thousands of integer variables and tens of thousands of constraints as long as the constraints and objective function are linear. The CPLEX solver uses a combination of branch and bound techniques, cutting plane algorithms, and heuristics, in an attempt to find the best feasible solution to an ILP within the minimum time. To use this advantage, we formulated Math Model #2 as an ILP within the Microsoft Visual C++ 2010 environment, and we used protocols from the IBM ILOG Concert Technology libraries to allow C++ to cooperate with the CPLEX solver.

5.2 CPLEX initialized with a feasible solution

We also combined the method for generating a feasible solution with the CPLEX solver, so that CPLEX can have a better chance to obtain a better result. The procedure of this method is shown in Figure 5-1. First, we generate a random feasible solution by the method proposed in Section 4.3. Then we collect the values of the *DYN*, *DAmt*, *SYN*, and *SAmt* variables and feed them as a start point for the CPLEX solver. After doing this, we expect the CPLEX solver to able to find an optimal solution more quickly because CPLEX will not waste time searching for an initial feasible solution or for solutions whose objective values are worse than the randomly generated initial feasible solution.

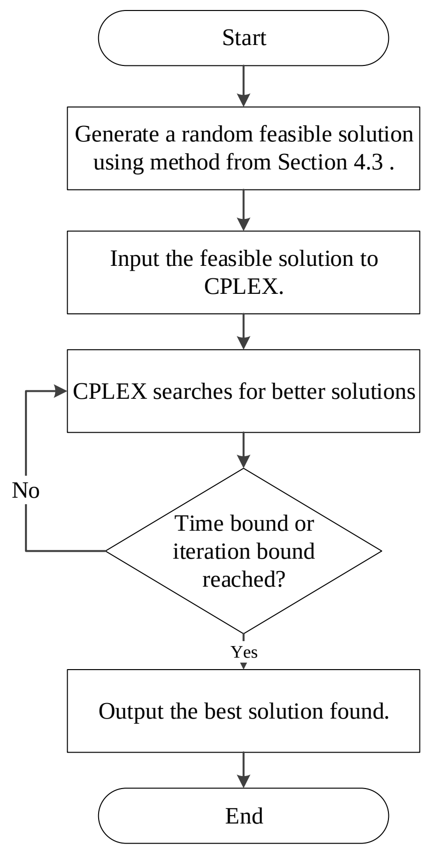

Figure 5-1. Integer programming procedure initialized with a feasible solution

5.3 Simulated annealing algorithm

The procedure of the simulated annealing (SA) algorithm developed in this thesis is presented as follows and the procedure is shown in Figure 5-2. An initial feasible solution is generated by the method described in Section 4.3. This solution is entirely specified by its (1) demand start points, (2) demand intervals, (3) supply start points, (4) supply intervals, and (5) supply subtraction epochs (see Sections 3-2 and 4-3). The elaboration of the simulated annealing algorithm proposed in this thesis includes the following:

(1) Initial Solution

To have an initial solution for the search procedure, we randomly generate an initial solution. The initial solution is guaranteed to be feasible because it follows the protocol from Section 4.3.

(2) Neighbor generation

Five neighborhoods are used in the global search. In the first neighborhood structure, one or more 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 one or more demanders. In the third neighborhood structure, one or more supply start points are changed to new random values between 1 and *T*. In the fourth neighborhood structure, the supply intervals are changed to new, random values for one or more suppliers. In the fifth neighborhood structure, the supply subtraction epochs are changed to new randomly selected values. The probability of using neighborhood #5 is (min{2*E*,20})% where *E*=*TotalS*-*TotalD*. The probability of using each of the neighborhoods #1-#4 is ((100-(probability of using neighborhood #5))/4)%. The procedure from Section 4.3 is then used to construct a neighboring solution, based on the demand start points, demand intervals, supply start points, supply intervals, and supply subtraction epochs of the neighboring solution.

(3) Acceptance probability

According to the principles of simulated annealing, when the neighboring solution is worse than the current solution, the probability of accepting the neighbor $Prob(accept) = e^{\frac{-\Delta}{P}}$, where P denotes the control temperature and Δ denotes the change in the objective value from the current to the neighboring solution. A worse neighbor will be accepted when $\gamma < Prob(accept)$, where γ is a uniformly distributed random variable between 0 and 1. Note that a neighboring solution is always accepted if its objective value is equal to or better than that of the current solution.

(4) Computation of temperatures

From the above discussion, SA exploits the temperature parameter P to control the diversification and intensification of the search path. Thus, it is important to choose the initial temperature wisely.

(5) Cooling factor

All SA algorithms use a cooling factor α to gradually lower the temperature. After every iteration of a SA algorithm, the current temperature P is lowered to the value αP where ($0<\alpha<1$). This gradual cooling is one feature that allows simulated annealing algorithms to be effective at finding near optimal solutions when dealing with large problems which contain numerous local optimums. The value of the cooling factor should be chosen wisely to optimize the performance of the SA algorithm.

(6) Terminating condition

The algorithm terminates when the maximum allowed CPU computation time expires.

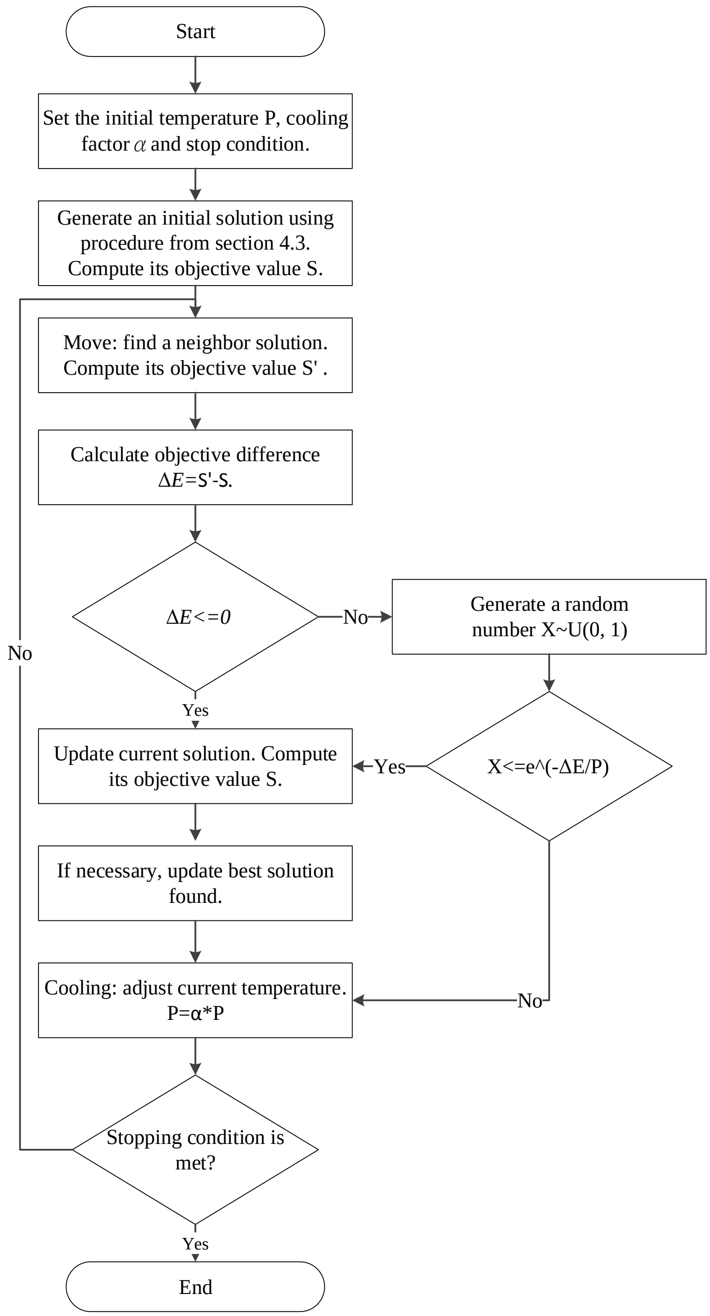

Figure 5-2. Simulated annealing algorithm procedure

5.4 Random algorithm

The final algorithm—the random algorithm—is a simple method that provides a benchmark for the SA algorithm. Figure 5-3 displays the procedure of the random algorithm. Basically, the method keeps randomly generating feasible solutions using the procedure from Section 4.3 until the time limit is reached. After that, the best solution found and its objective values are outputted.

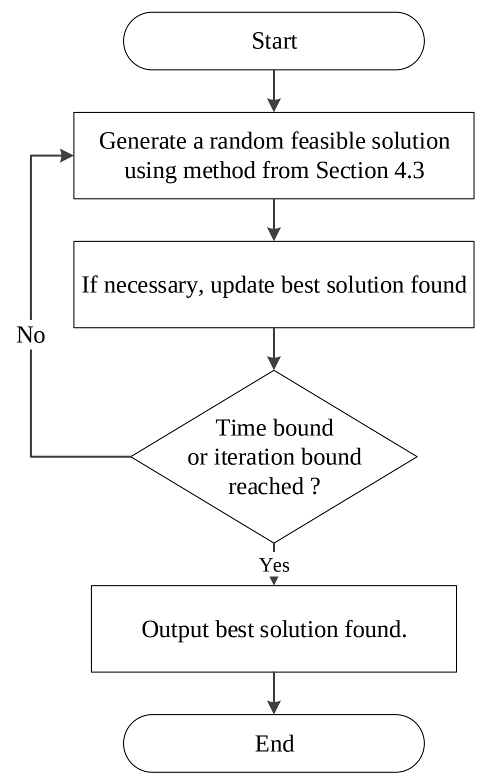

Figure 5-3. Random algorithm procedure

CHAPTER 6: COMPUTATIONAL RESULTS

This chapter presents and discusses our experiments that compare the performance of the four methods described in Chapter 5 on several problem instances. This chapter is organized as follows. In Section 6.1, we introduce the method for generating the problem instances that are considered in the experiments. Section 6.2 describes the software, hardware, and algorithm stopping condition used for experimentation. Section 6.3 describes our efforts to calibrate the SA algorithm. The purpose of the calibration is to decide the values of the initial temperature P and cooling factor α—the two major factors that impact the searching ability. Section 6.4 and 6.5 present and discuss the experimental results concerning two types of problem instances. Section 6.4 presents the results for the easy problem instances. Section 6.5 presents the results for the hard problem instances in which we force $TotalS - TotalD \leq 10$.

6.1 Generating problem instances

To be more comprehensive to this research, we create not only small size problems but also large size problems. In this manner, we can compare each method's ability to solve small size problems versus large problems.

Table 6-1 shows the parameter value ranges used for generating the problem instances. In all instances, D and S equal 2, 6, 20, or 60. T equals 10, 30, or 100. In each instance, the demand quantities, demand frequencies, supply quantities and supply frequencies are random variables from the discrete uniform (DU) distribution within the ranges displayed in Table 6-1.

Table 6-1. Parameter value ranges for the experiments

Parameter	Range of possible values	# of possible values
D, *S*	2, 6, 20, 60	4
T	10, 30, 100	3
DQ_d	DU(1, 9)	9
DF_d	DU(2, 9)	8
SQ_s	DU(1, 9)	9
SF_s	DU(2, 9)	8

Table 6-2. Instance categories considered in the experiments

Number of demanders (*D*) and suppliers (*S*)	Length of cycle time (*T*)	Instance category
2	10	d02s02t010
2	30	d02s02t030
2	100	d02s02t100
6	10	d06s06t010
6	30	d06s06t030
6	100	d06s06t100
20	10	d20s20t010
20	30	d20s20t030
20	100	d20s20t100
60	10	d60s60t010
60	30	d60s60t030
60	100	d60s60t100

Table 6-2 displays the problem sizes (i.e. instance categories) that are considered. A total of 12 instance categories are considered, corresponding to all possible combinations for the number of demanders and suppliers—2, 6, 20, and 60—and the length of the cycle—10 days, 30 days, and 100 days. Furthermore, 10 instances are considered in each category. Thus, a total of 120 instances are considered in the experiments.

Two instance difficulty levels are considered in the experiments. For the easy instances, no particular stipulations are placed on the instances other than the requirement that they be feasible. In the hard instances, we stipulate that the instances be feasible and that $TotalS - TotalD \leq 10$. This gives the decision maker less flexibility regarding supply quantities and supply subtraction epochs.

6.2 Software settings, hardware settings, and termination criteria

All the experiments are coded using the Microsoft Visual C++ 2010 professional compiler and IBM ILOG CPLEX 12.5 under the Windows 7 operating system and are executed on a personal computer equipped with an 8-core Intel i7-4770 3.4 GHz CPU with 16 GB of RAM. All algorithms are required to terminate after 60 seconds of computation time have elapsed.

6.3 Simulated annealing algorithm settings

In this subchapter, we attempt to find the best settings for the SA algorithm. The two main factors in the simulated annealing algorithm are the temperature and cooling factor. In preliminary experiments, we found out that the SA algorithm can generate millions of neighboring solutions within the 60 second time limit. As a result, we set the cooling factor to 0.999 so the temperature will not freeze too early. If the temperature drops too rapidly, it will increase the chance that the algorithm will become stuck in a local optimal solution.

The four values considered for the temperature factor are shown in Table 6-3. Preliminary experiments will compare the performance of these four options. After comparison, we will choose the best setting to use in our final experiments.

Table 6-3. Simulated annealing algorithm parameter settings

Level / factor	1	2	3	4
Temperature (P)	1000	100	10	1
Cooling factor (α)	0.999			

Table 6-4. Simulated annealing results with P=1000 and α=0.999

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t010	OBJ1	3	0	0	12	1	4	12	4	5	0	4.1
	OBJ2	1	0	0	3	1	1	3	4	4	0	1.7
d02s02t030	OBJ1	23	16	16	65	90	0	10	12	55	46	33.3
	OBJ2	3	4	2	5	6	0	2	1	6	5	3.4
d02s02t100	OBJ1	310	113	123	380	258	95	168	408	101	204	216
	OBJ2	6	3	3	9	6	3	8	9	3	5	5.5
d06s06t010	OBJ1	2	3	2	7	4	1	1	2	0	2	2.4
	OBJ2	0	1	0	3	1	1	1	0	0	0	0.7
d06s06t030	OBJ1	32	21	64	37	19	16	50	41	43	46	36.9
	OBJ2	3	2	6	4	2	1	3	4	2	3	3
d06s06t100	OBJ1	344	194	351	311	271	315	359	373	369	480	336.7
	OBJ2	8	5	10	8	6	7	9	8	8	10	7.9
d20s20t010	OBJ1	2	0	0	0	0	0	0	0	0	6	0.8
	OBJ2	0	0	0	0	0	0	0	0	0	0	0
d20s20t030	OBJ1	73	37	40	46	53	38	65	40	35	37	46.4
	OBJ2	2	2	2	3	3	4	4	3	4	2	2.9
d20s20t100	OBJ1	337	315	416	429	486	288	417	346	361	428	382.3
	OBJ2	9	8	11	10	9	9	9	8	11	10	9.4
d60s60t010	OBJ1	5	0	1	0	1	0	0	1	2	2	1.2
	OBJ2	0	1	0	0	1	0	0	0	0	0	0.2
d60s60t030	OBJ1	34	51	27	68	20	60	32	37	36	36	40.1
	OBJ2	2	3	3	3	3	3	2	2	3	3	2.7
d60s60t100	OBJ1	398	424	295	328	428	350	483	288	324	424	374.2
	OBJ2	12	14	13	13	14	12	15	12	12	12	12.9

Table 6-4 shows the results for the SA algorithm on the 120 easy problem instances when *P*=1000 and α=0. 999. The first column shows the instance category. For example, d02s02t010 means this problem has 2 demanders and 2 suppliers with cycle length of 10 days. The second column shows which objective is being considered—objective 1 or objective 2. Columns 0 to 9 relate to the ten individual instances within each instance category. The last column shows the average objective value of each problem size. There are 12 rows in the table. Each row represents a problem size. The values in the table are the best objective values found by the algorithm within the 60 second time limit.

Table 6-5 shows the results for the SA algorithm on the 120 easy problem instances when *P*=100 and α=0. 999. In this setting most of the results are similar to the setting *P*=1000 and α=0.999. However, some instances' objective values are worse than setting *P*=1000. The worse situation can be explained by the different initial temperature. In larger problems such as d20s20t100 and d60s60t100, a lower initial temperature means that the procedure of searching will freeze earlier. That is, there is a higher chance that the search will become trapped in a local optimal solution. This helps to explain why the results for category d20s20t100 with temperature 100 are worse than with temperature 1000 by 13.7%.

Table 6-6 shows the results for the SA algorithm on the 120 easy problem instances when *P*=10 and α=0.999. In this setting we see that some instances' objective values are different or worse compared to the setting *P*=1000 and α=0.999. The worse situation can be explained by the low initial temperature. In larger problems, such as d20s20t100 and d60s60t100, an initial temperature of 10 means that the procedure of searching will freeze earlier then when *P*=1000 and *P*=100. In that case, the chance that the search procedure will become trapped in a local optimum is higher.

Table 6-5. Simulated annealing results with *P*=100 and α=0.999

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t010	OBJ1	3	1	0	12	0	4	10	4	5	0	3.9
	OBJ2	1	0	0	3	0	1	3	4	4	0	1.6
d02s02t030	OBJ1	18	26	22	65	90	0	9	12	55	46	34.3
	OBJ2	3	4	2	5	6	0	2	1	6	5	3.4
d02s02t100	OBJ1	262	126	106	371	252	115	170	352	91	221	206.6
	OBJ2	7	3	3	9	6	3	8	9	3	6	5.7
d06s06t010	OBJ1	0	3	0	5	4	1	2	3	0	2	2
	OBJ2	0	1	0	1	1	1	0	1	0	0	0.5
d06s06t030	OBJ1	22	23	66	38	16	29	50	41	60	28	37.3
	OBJ2	3	2	5	3	2	1	4	4	1	3	2.8
d06s06t100	OBJ1	311	186	447	373	272	332	370	322	328	428	336.9
	OBJ2	8	5	10	7	6	7	9	7	8	9	7.6
d20s20t010	OBJ1	1	0	0	0	0	0	0	1	1	0	0.3
	OBJ2	0	0	0	0	0	0	0	0	0	0	0
d20s20t030	OBJ1	61	38	44	40	37	42	44	37	41	45	42.9
	OBJ2	2	2	3	3	3	3	3	3	3	2	2.7
d20s20t100	OBJ1	386	413	366	498	427	461	443	399	502	453	434.8
	OBJ2	7	9	10	10	10	9	11	8	11	9	9.4
d60s60t010	OBJ1	0	0	2	0	4	0	0	3	0	0	0.9
	OBJ2	0	0	0	0	0	0	0	0	0	0	0
d60s60t030	OBJ1	53	29	54	55	26	24	62	19	27	32	38.1
	OBJ2	2	3	3	2	3	4	3	3	3	3	2.9
d60s60t100	OBJ1	484	471	299	364	414	374	421	319	434	335	391.5
	OBJ2	12	10	10	15	12	12	11	13	11	12	11.8

Table 6-6. Simulated annealing results with P=10 and α=0.999

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t010	OBJ1	3	0	0	16	2	5	10	4	5	0	4.5
	OBJ2	1	1	0	3	0	1	3	4	4	0	1.7
d02s02t030	OBJ1	18	16	13	65	90	0	14	13	55	50	33.4
	OBJ2	3	4	2	5	6	0	2	1	6	5	3.4
d02s02t100	OBJ1	215	129	131	368	259	100	239	432	95	238	220.6
	OBJ2	6	3	3	9	6	3	8	9	3	6	5.6
d06s06t010	OBJ1	0	3	0	12	8	1	5	4	1	0	3.4
	OBJ2	0	1	0	2	1	1	0	0	0	0	0.5
d06s06t030	OBJ1	35	32	62	47	23	34	88	46	38	47	45.2
	OBJ2	2	2	5	4	2	2	5	5	1	3	3.1
d06s06t100	OBJ1	378	238	416	357	318	283	467	429	374	465	372.5
	OBJ2	8	5	10	7	6	7	9	6	8	11	7.7
d20s20t010	OBJ1	0	3	0	0	0	0	1	0	0	0	0.4
	OBJ2	0	0	0	0	0	0	0	0	0	0	0
d20s20t030	OBJ1	43	48	57	55	35	41	90	26	46	43	48.4
	OBJ2	3	2	3	3	2	2	2	2	3	2	2.4
d20s20t100	OBJ1	462	518	479	478	552	411	375	327	562	441	460.5
	OBJ2	10	9	9	8	10	8	9	8	11	9	9.1
d60s60t010	OBJ1	1	0	1	1	0	0	0	0	0	0	0.3
	OBJ2	0	0	0	1	0	0	0	0	0	0	0.1
d60s60t030	OBJ1	41	107	73	47	44	68	16	48	47	25	51.6
	OBJ2	3	3	4	3	3	3	3	2	3	3	3
d60s60t100	OBJ1	629	541	511	636	545	444	604	437	480	536	536.3
	OBJ2	13	12	12	14	12	13	13	12	13	12	12.6

Table 6-7 shows the results for the SA algorithm on the 120 easy problem instances when P=1 and α=0.999. In this setting we see that most instances' objective values are worse than when of P=1000, P=100, and P=10. The worse situation can be explained by the low initial temperature. In most cases, an initial temperature of 1 means that the procedure of searching will freeze earlier than when P=1000, P=100, and P=10. In that case, the chance that the search procedure will become trapped in a local optimum is higher.

Table 6-7. Simulated annealing results with P=1 and α=0.999

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t010	OBJ1	4	1	4	12	4	7	20	6	5	2	6.5
	OBJ2	1	1	0	3	1	1	3	4	4	0	1.8
d02s02t030	OBJ1	38	27	14	80	90	0	17	12	60	63	40.1
	OBJ2	3	4	2	5	6	1	2	1	6	5	3.5
d02s02t100	OBJ1	274	116	154	375	313	134	202	386	110	225	228.9
	OBJ2	6	3	3	9	6	3	8	9	3	6	5.6
d06s06t010	OBJ1	0	3	12	19	7	6	1	11	0	14	7.3
	OBJ2	0	1	1	2	0	1	0	1	0	0	0.6
d06s06t030	OBJ1	29	28	56	64	17	84	63	55	33	42	47.1
	OBJ2	3	2	6	4	2	2	3	4	3	3	3.2
d06s06t100	OBJ1	429	220	436	398	309	399	455	440	429	388	390.3
	OBJ2	8	5	10	7	7	8	9	7	9	10	8
d20s20t010	OBJ1	0	0	0	2	0	2	2	0	0	2	0.8
	OBJ2	0	0	0	0	0	0	0	0	0	0	0
d20s20t030	OBJ1	39	80	56	63	54	70	83	50	59	87	64.1
	OBJ2	3	3	2	2	2	2	2	2	3	2	2.3
d20s20t100	OBJ1	503	572	596	396	438	377	635	396	638	390	494.1
	OBJ2	8	7	11	8	10	9	8	8	11	10	9
d60s60t010	OBJ1	0	0	0	1	0	0	0	0	0	1	0.2
	OBJ2	0	0	0	0	0	0	0	0	0	0	0
d60s60t030	OBJ1	67	21	78	43	38	35	34	54	36	58	46.4
	OBJ2	3	3	2	3	2	2	2	2	3	3	2.5
d60s60t100	OBJ1	420	499	455	528	798	731	502	494	728	835	599
	OBJ2	13	11	12	15	15	12	16	12	12	11	12.9

Table 6-8 shows average total number of iterations, average iteration number when the best solution is found, and the average number of neighboring solutions accepted for the experiments from Table 6-4 concerning objective 1 (P=1000 and α=0. 999). In this table, we can see that the average total number of iterations is decreasing as the problem size increases. For example, for problem sizes d02s02t010, d06s06t010, d20s20t010, and d60s60t010, there is a trend that when the number of demanders and suppliers goes up the average total number of iterations that can be completed within the time limit goes down. On the other hand, when only the cycle length increases, average total number of iterations will decrease. These observations agree with intuition.

The other fact we can find out in Table 6-8 is that when the SA algorithm solves small problems, the best solution can be found earlier in the entire searching process than for large problems. For example, for problem sizes d02s02t010, d02s02t030, and d02s02t100 "Avg. best iteration" is increasing with the cycle length. Note in column "Avg. # accepted" that neighboring solutions are accepted for about 20% of the iterations for most problem sizes.

Table 6-8. Detailed simulated annealing results with P=1000 and α=0.999

Problem size	Avg. total iteration	Avg. best iteration	Avg. # accepted
d02s02t010	3651886	6095	963330
d02s02t030	2808722	153028	780598
d02s02t100	1543960	210564	262906
d06s06t010	2809549	18404	823578
d06s06t030	1669276	146072	461401
d06s06t100	648196	142713	115775
d20s20t010	1659768	89852	621086
d20s20t030	657810	206895	180783
d20s20t100	196335	93290	43578
d60s60t010	760052	126158	476076
d60s60t030	239104	87170	63956
d60s60t100	51281	30673	15793

Table 6-9 shows the detailed results for the experiments from Table 6-5 (objective 1 only) where P=100 and α=0.999. Here again, we see that the average total number of iterations is decreasing as the problem size increases. For example, for problem sizes d02s02t010, d06s06t010, d20s20t010, and d60s60t010, there is a trend that when the number of demanders and suppliers goes up the average total number of iterations goes down. On the other hand, when only the cycle length increases, the average total number of iterations will decrease.

Table 6-9. Detailed simulated annealing results with P=100 and α=0.999

Problem size	Avg. total iteration	Avg. best iteration	Avg. # accepted
d02s02t010	3647096	5896	896280
d02s02t030	2811906	50906	832105
d02s02t100	1547149	154081	262175
d06s06t010	2807321	88324	824956
d06s06t030	1674763	342281	448378
d06s06t100	646671	281772	114327
d20s20t010	1620074	53207	478701
d20s20t030	659404	213317	178160
d20s20t100	198025	117554	42966
d60s60t010	750880	179915	456011
d60s60t030	252475	120095	79605
d60s60t100	51812	37694	14188

Table 6-10. Detailed simulated annealing results with P=10 and α=0.999

Problem size	Avg. total iteration	Avg. best iteration	Avg. # accepted
d02s02t010	3580073	3551	895835
d02s02t030	2757993	202027	799997
d02s02t100	1533605	271400	258562
d06s06t010	2788498	36275	789727
d06s06t030	1670562	369609	463702
d06s06t100	646049	257741	110857
d20s20t010	1655042	80868	574303
d20s20t030	656069	233120	174794
d20s20t100	196818	111793	40695
d60s60t010	750423	115795	449765
d60s60t030	239461	81336	67106
d60s60t100	51044	31815	12213

Table 6-10 displays the detailed results for the experiments from Table 6-6 (objective 1 only) where *P*=10 and α=0. 999. Here we can see that the average total number of iterations is also decreasing as the problem size goes up.

Table 6-11 shows the detailed results for the experiments from Table 6-6 (objective 1 only) where *P*=1 and α=0. 999. Here we can see that the average number of total iterations is also decreasing as the problem size goes up.

Table 6-11. Detailed simulated annealing results with *P*=1 and α=0.999

Problem size	Avg. total iteration	Avg. best iteration	Avg. # accepted
d02s02t010	3628787	1641	954616
d02s02t030	2804909	176304	752671
d02s02t100	1539674	94427	242842
d06s06t010	2799561	157964	818074
d06s06t030	1668477	253513	460714
d06s06t100	645475	111536	111453
d20s20t010	1618086	119283	506098
d20s20t030	656710	263980	174441
d20s20t100	196399	107478	39521
d60s60t010	734746	127221	297833
d60s60t030	220223	88704	51547
d60s60t100	51131	30508	11105

Figure 6-1 summarizes the performance of the four temperature levels for objective 1. In most cases, the setting *P*=1000 and α=0.999 has better performance than other settings. This is especially true for the large problem sizes such as d20s20t100 and d60s60t100.

Figure 6-2 summarizes the performance of the four temperature levels for objective 2. In this figure we observe no significant difference in performance between the options.

Based on these results, we decide to use the settings *P*=1000 and α=0.999 for comparison with the other three algorithms in Sections 6.4 and 6.5.

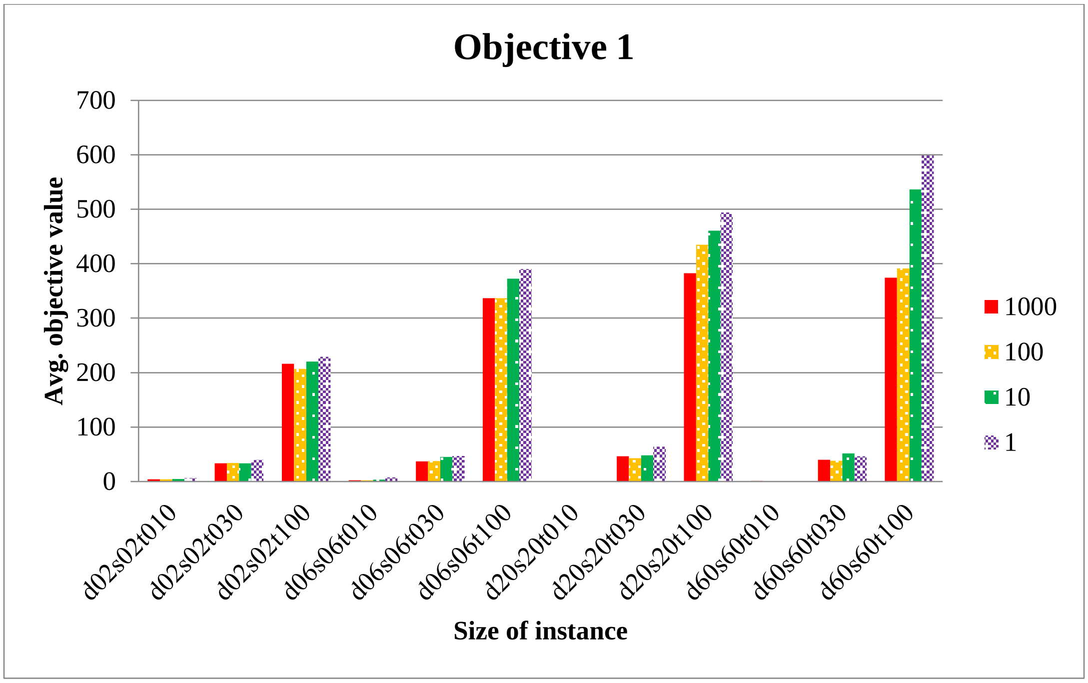

Figure 6-1. Summary of results for simulated annealing algorithms (objective 1)

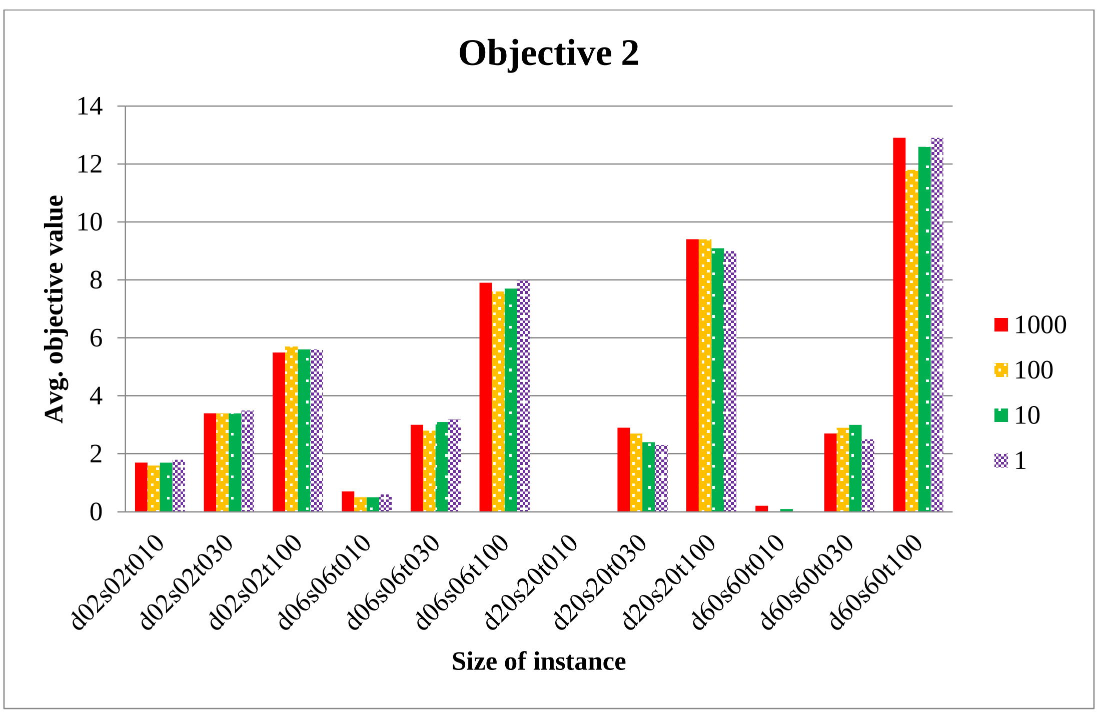

Figure 6-2. Summary of results for simulated annealing algorithms (objective 2)

6.4 Results for easy problem instances

Table 6-12 shows the results when CPLEX is called to solve Math Model #2 for the 120 easy problem instances. The first column indicates the problem size. Experiments consider ten randomly generated problem instances for each problem size.

Table 6-12. Experimental results for CPLEX without an initial feasible solution (easy instances)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t10	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
d02s02t30	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
d06s06t10	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
d06s06t30	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**	**0**	81	**0**	8.1
	OBJ2	**0**	**0**	**0**	**0**	**0**	2	**0**	3	5	1	1.1
d20s20t10	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
d20s20t30	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
d20s20t100	OBJ1	150	210	209	**0**	**0**	662	**0**	355	**0**	474	206
	OBJ2	N/A	18	4496	**0**	**0**	25	**0**	19	**0**	23	-
d60s60t10	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
d60s60t30	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	3066	N/A	356	N/A	N/A	130	N/A	N/A	N/A	-
	OBJ2	N/A	**0**	N/A	N/A	N/A	N/A	**0**	N/A	N/A	N/A	-

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

Each problem instance is considered twice: using objective 1 and using objective 2. Columns 0 to 9 relate to the ten individual instances within each instance category. The last column shows the average optimal value across all instances for each problem size and objective. Numbers in bold denote provably optimal values. The term "N/A" means that CPLEX was unable to identify a feasible solution within the 1 minute time limit.

Table 6-13. Experimental results for CPLEX with an initial feasible solution (easy instances)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t10	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
d02s02t30	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
d06s06t10	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
d06s06t30	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**	23	90	14	12.7
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	4	8	1	1.3
d20s20t10	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
d20s20t30	OBJ1	**0**	**0**	**0**	16	12	**0**	**0**	**0**	**0**	13	4.1
	OBJ2	1	**0**	1	2	3	**0**	**0**	**0**	**0**	4	1.1
d20s20t100	OBJ1	715	452	342	**0**	**0**	725	209	302	**0**	385	313.0
	OBJ2	23	29	6	**0**	**0**	39	0	14	**0**	24	13.5
d60s60t10	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
d60s60t30	OBJ1	**0**	**0**	**0**	15	**0**	**0**	**0**	2	**0**	**0**	1.7
	OBJ2	**0**	**0**	**0**	58	**0**	**0**	**0**	**0**	**0**	**0**	5.8
d60s60t100	OBJ1	4617	464	1592	3945	1148	1522	6524	1407	4759	755	2673.3
	OBJ2	105	43	70	80	159	82	**0**	47	117	35	73.8

Bold: Optimal solution

Table 6-13 displays the results on the easy problem instance for the second solution method—CPLEX initialized with a feasible solution. The first column indicates the problems size. Columns 0 to 9 relate to the ten individual instances within each instance category. The last column shows the average optimal value across all instances for each problem size and objective.

Table 6-14 shows the results on the easy instances for the third solution method—the simulated annealing algorithm with P=1000 and α=0.999. This table is identical to Table 6-4.

Table 6-14. Experimental results for the simulated annealing algorithm (easy instances; same as Table 6-4)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d02s02t010	OBJ1	3	**0**	**0**	12	1	**4**	12	4	**5**	**0**	4.1
	OBJ2	**1**	**0**	**0**	**3**	1	**1**	**3**	4	**4**	**0**	1.7
d02s02t030	OBJ1	23	16	16	**65**	**90**	**0**	10	12	**55**	46	33.3
	OBJ2	3	4	**2**	**5**	**6**	**0**	2	1	**6**	**5**	3.4
d02s02t100	OBJ1	310	113	123	380	258	95	168	408	101	204	216
	OBJ2	6	3	3	9	6	3	**8**	9	3	**5**	5.5
d06s06t010	OBJ1	2	3	2	7	4	1	1	2	**0**	2	2.4
	OBJ2	**0**	**1**	**0**	3	1	1	1	**0**	**0**	**0**	0.7
d06s06t030	OBJ1	32	21	64	37	19	16	50	41	43	46	36.9
	OBJ2	3	2	6	4	2	1	3	4	2	3	3
d06s06t100	OBJ1	344	194	351	311	271	315	359	373	369	480	336.7
	OBJ2	8	5	10	8	6	7	9	8	8	10	7.9
d20s20t010	OBJ1	2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	6	0.8
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
d20s20t030	OBJ1	73	37	40	46	53	38	65	40	35	37	46.4
	OBJ2	2	2	2	3	3	4	4	3	4	2	2.9
d20s20t100	OBJ1	337	315	416	429	486	288	417	346	361	428	382.3
	OBJ2	9	8	11	10	9	9	9	8	11	10	9.4
d60s60t010	OBJ1	5	**0**	1	**0**	1	**0**	**0**	1	2	2	1.2
	OBJ2	**0**	1	**0**	**0**	1	**0**	**0**	**0**	**0**	**0**	0.2
d60s60t030	OBJ1	34	51	27	68	20	60	32	37	36	36	40.1
	OBJ2	2	3	3	3	3	3	2	2	3	3	2.7
d60s60t100	OBJ1	398	424	295	328	428	350	483	288	324	424	374.2
	OBJ2	12	14	13	13	14	12	15	12	12	12	12.9

Bold: Optimal solution

Table 6-15 displays the results on the easy instances for the final solution method—the random algorithm.

Table 6-15. Experimental results for the random algorithm (easy instances)

Problem size	Objective	Instance 0	1	2	3	4	5	6	7	8	9	Avg.
d02s02t10	OBJ1	**2**	**0**	**0**	13	**0**	**4**	**10**	4	**5**	**0**	3.8
	OBJ2	**1**	**0**	**0**	4	**0**	**1**	**3**	2	**4**	**0**	1.5
d02s02t30	OBJ1	24	34	34	72	80	6	39	23	**55**	55	42.2
	OBJ2	2	4	4	**5**	**6**	1	4	2	**6**	6	4.0
d02s02t100	OBJ1	342	203	212	452	332	145	253	402	195	280	281.6
	OBJ2	7	5	6	10	7	4	**8**	9	5	6	6.7
d06s06t10	OBJ1	8	9	6	9	6	8	8	2	10	6	7.2
	OBJ2	2	2	2	3	2	2	2	1	2	2	2.0
d06s06t30	OBJ1	75	53	102	110	61	75	103	82	82	96	83.9
	OBJ2	6	5	9	9	5	6	8	7	6	7	6.8
d06s06t100	OBJ1	610	449	656	604	444	562	606	540	665	776	591.2
	OBJ2	13	10	16	14	10	13	15	15	14	18	13.8
d20s20t10	OBJ1	21	20	18	19	14	19	29	14	12	16	18.2
	OBJ2	5	6	5	6	3	5	7	4	3	3	4.7
d20s20t30	OBJ1	187	177	195	195	178	174	191	187	177	177	183.8
	OBJ2	15	14	13	14	14	16	15	14	16	12	14.3
d20s20t100	OBJ1	1076	1126	1227	1298	1339	1080	1206	1001	1348	1213	1191.4
	OBJ2	22	26	28	29	29	25	27	23	29	26	26.4
d60s60t10	OBJ1	47	40	45	52	42	48	48	52	46	44	46.4
	OBJ2	10	9	10	10	10	13	9	12	9	11	10.3
d60s60t30	OBJ1	333	374	394	295	399	357	333	330	339	305	345.9
	OBJ2	25	28	31	28	29	29	28	29	26	24	27.7
d60s60t100	OBJ1	2198	1963	2022	2117	2007	1878	2291	1923	1959	2037	2039.5
	OBJ2	48	40	46	49	48	41	46	46	48	46	45.8

Bold: Optimal solution

Table 6-16 compares the average number of iterations executed by the random algorithm and the simulated annealing algorithm. As the table shows, the number of iterations for the algorithm is higher than for the simulated annealing algorithm. From this information, we determine that the simulated annealing algorithm's superiority over the random algorithm is not due to the total number of iterations it considers, but rather due to its superior searching ability.

Table 6-16. Iteration comparison of random and simulated annealing algorithms

Problem size	Random	SA-1000	*RPD
	Avg. total # iterations	Avg. total # iterations	%
d02s02t010	4962797	3651886	26%
d02s02t030	3907395	2808722	28%
d02s02t100	2265827	1543960	32%
d06s06t010	3914124	2809549	28%
d06s06t030	2449035	1669276	32%
d06s06t100	990053	648196	35%
d20s20t010	2120100	1659768	22%
d20s20t030	961769	657810	32%
d20s20t100	292531	196335	33%
d60s60t010	889318	760052	15%
d60s60t030	316083	239104	24%
d60s60t100	68811	51281	25%

(*Relative Percent Deviation, RPD)

Table 6-17 summarizes the performance of the four methods. It shows the average objective value achieved by each method for each problem size and objective. Note that the CPLEX method has some "-" in the table. This means that the CPLEX solver could not find a feasible solution within given time limit (60 seconds) for one or more instances in the category.

The overall performance of the four methods is as follows. Interestingly, the first method—pure CPLEX—generally finds the lowest objective value among all the methods. Indeed, as shown in Table 6-12, CPLEX finds an optimal solution for the majority of the 120 easy problem instances that are considered. Interestingly, we can observe that even if we give a feasible solution to CPLEX as a start point, there is a decent chance that it will lead to a worse result than using pure CPLEX. However, when there are 20 or 60 demanders and suppliers and a large cycle length, CPLEX sometimes cannot find a feasible solution within 60 seconds. For such cases, it is better to use the second method—CPLEX initialized with a feasible solution—to generate a feasible solution for CPLEX as an initial start point.

It is noteworthy that the simulated annealing algorithm can find much better solutions than either CPLEX method when there are 60 demanders and 60 suppliers with a cycle length of 100. However, the SA algorithm generally does not perform as well as the CPLEX-based methods on the other problem instances. Nevertheless, the SA algorithm significantly outperforms the random algorithm for the vast majority of problem sizes.

Table 6-17. Overall experimental results (easy instances)

Problem size	Objective	CPLEX w/o Initial Feas. Soln.	CPLEX w/ Initial Feas. Soln.	SA	Random
d2s2t10	Objective 1	**3.1**	**3.1**	4.1	3.7
	Objective 2	**1.2**	**1.2**	1.7	1.3
d2s2t30	Objective 1	**25.2**	**25.2**	33.3	36.6
	Objective 2	**2.4**	**2.4**	3.4	3.7
d2s2t100	Objective 1	**98.8**	**98.8**	216	263.3
	Objective 2	**3.3**	**3.3**	5.5	6.6
d6s6t10	Objective 1	**0.1**	**0.1**	2.4	4.4
	Objective 2	**0.1**	**0.1**	0.7	1.5
d6s6t30	Objective 1	**3.9**	**3.9**	36.9	67.9
	Objective 2	**0.7**	**0.7**	3	6.2
d6s6t100	Objective 1	**8.1**	12.7	336.7	532.9
	Objective 2	**1.1**	1.3	7.9	12.7
d20s20t10	Objective 1	**0**	**0**	0.8	15
	Objective 2	**0**	**0**	**0**	3.5
d20s20t30	Objective 1	4.5	**4.1**	46.4	154.4
	Objective 2	**1**	1.1	2.9	12.7
d20s20t100	Objective 1	**206**	313	382.3	1065.5
	Objective 2	-	13.5	**9.4**	24.2
d60s60t10	Objective 1	**0**	**0**	1.2	29.8
	Objective 2	**0**	**0**	0.2	7.4
d60s60t30	Objective 1	**0.3**	1.7	40.1	300.4
	Objective 2	-	5.8	**2.7**	23.4
d60s60t100	Objective 1	-	2673.3	**374.2**	1870.1
	Objective 2	-	73.8	**12.9**	42.1

Bold: Best performance among the four methods

Figures 6-3 through 6-6 show the overall results to a greater degree of aggregation than Table 6-17. Figure 6-3 shows the individual impact of the solution method (left) and problem size (right) on the best value that is found for objective 1. The results for the first method—pure CPLEX—are not included because it cannot find a feasible solution for some instances. The second method—CPLEX with an initial feasible solution—generally has better performance than other methods but it cannot find good feasible solutions for the d60s60t100 instances so its performance appears slightly worse than SA algorithm in the figure. Regarding the right side of the figure, note that the objective value goes up as the number of demanders, suppliers, and/or cycle length increases.

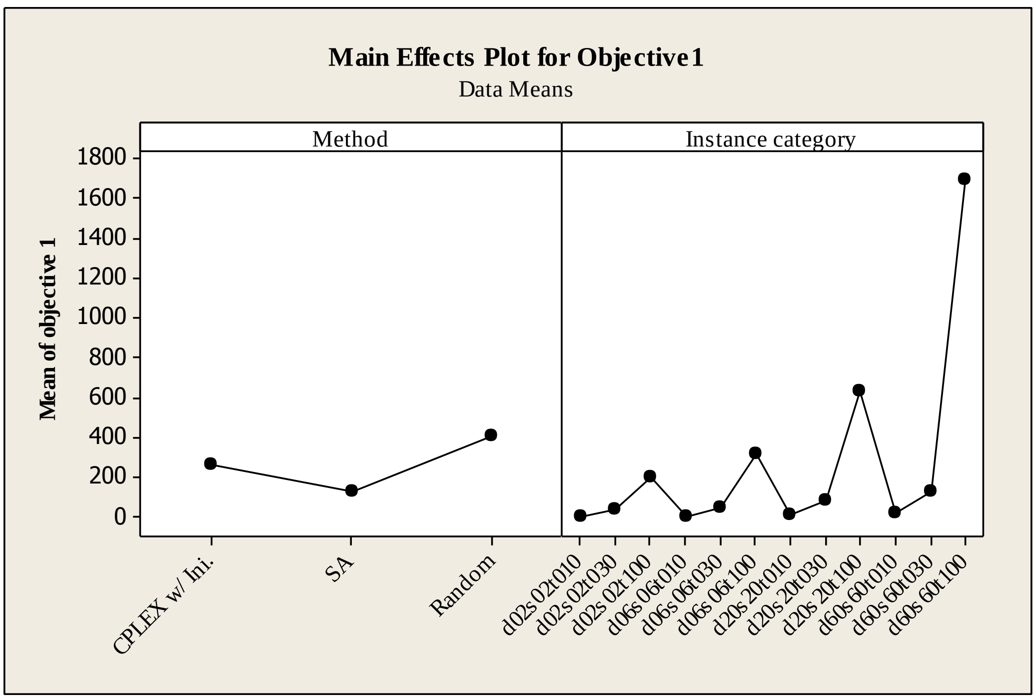

Figure 6-3. Avg. value of objective 1 by method (left) and by problem size (right) (easy instances)

Figure 6-4 illustrates the combined impact of the solution method and problem size on the best value that is found for objective 1. Here we see that when the number of demanders and suppliers goes up, the objective value will also go up. The same situation happens regarding the length of the cycle.

Figure 6-5 shows the individual impact of the solution method (left) and problem size (right) on the best value that is found for objective 2. The results for the first method—pure CPLEX—are not included because it cannot find a feasible solution for some instances. The second method—CPLEX with an initial feasible solution—generally has better performance than other methods but it cannot find good feasible solutions for the d60s60t100 instances so its performance appears slightly worse than SA algorithm in the figure. Regarding the right side of the figure, note that the objective value goes up as the number of demanders, suppliers, and/or cycle length increases.

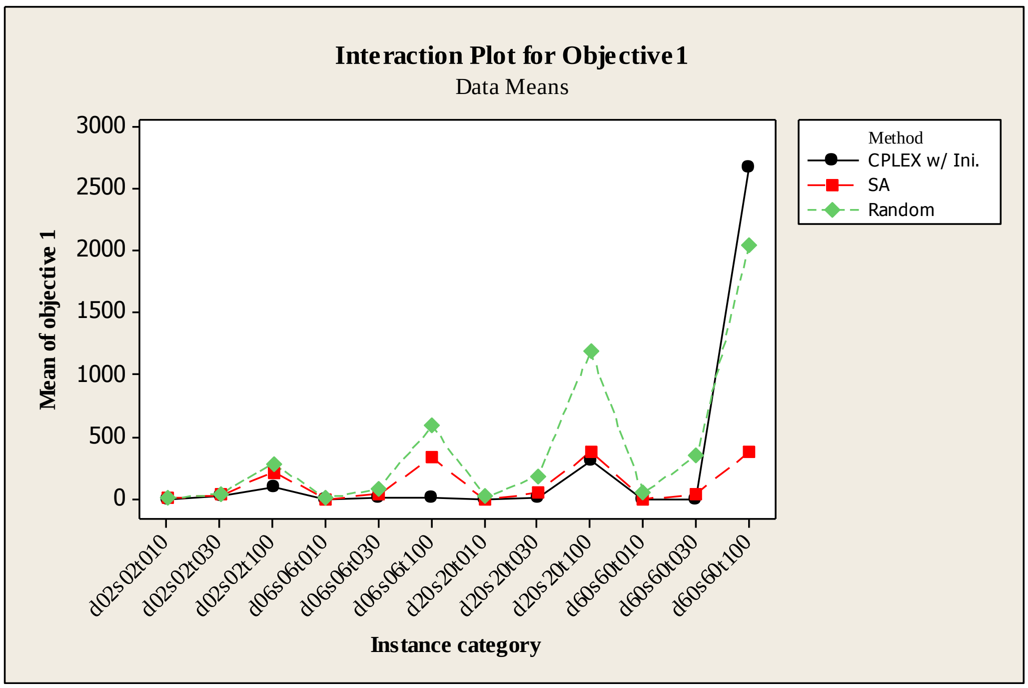

Figure 6-4. Avg. value of objective 1 achieved for each combination of method and problem size (easy instances)

Figure 6-6 illustrates the combined impact of the solution method and problem size on the best value that is found for objective 2. Here we see that when the number of demanders and suppliers goes up, the objective value will also go up. The same situation happens regarding the length of the cycle.

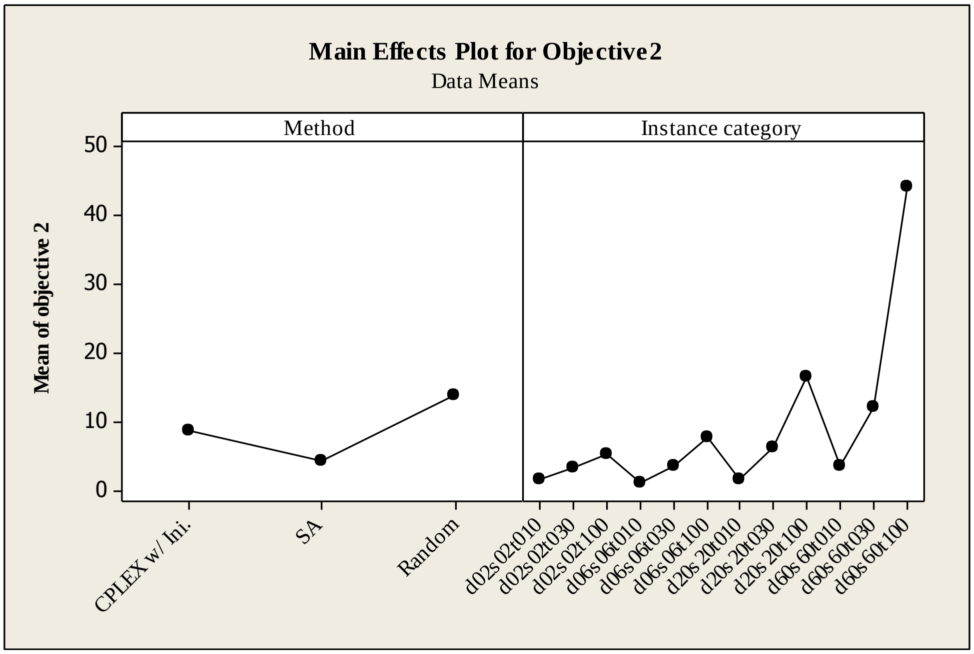

Figure 6-5. Avg. value of objective 2 by method (left) and by problem size (right) (easy instances)

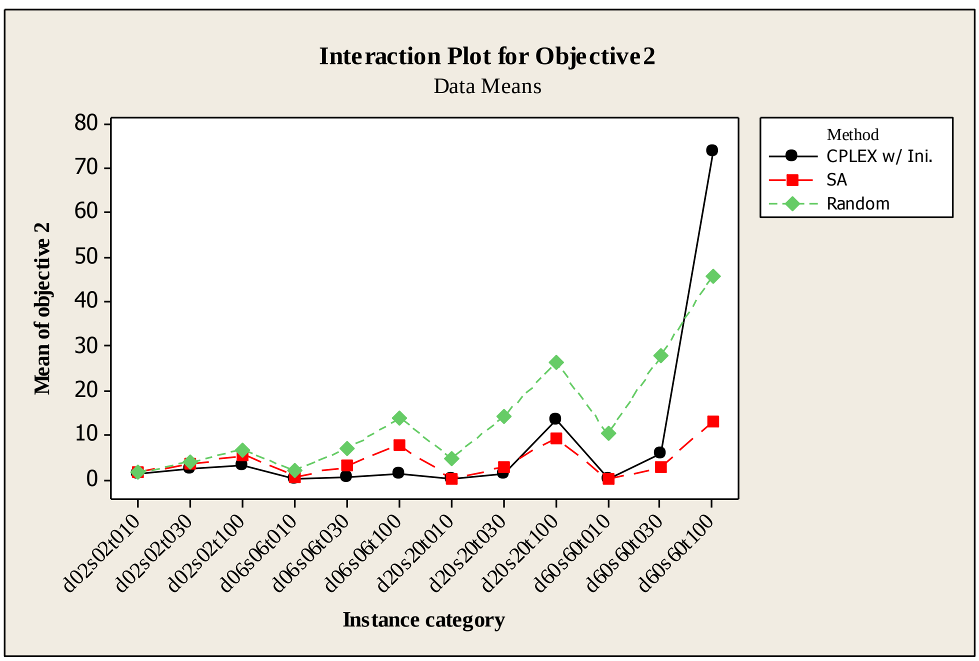

Figure 6-6. Avg. value of objective 2 achieved for each combination of method and problem size (easy instances)

6.5 Results for hard problem instances

The results from Section 6.4 show that CPLEX performs quite well on most problems but CPLEX is not doing so well on the largest (i.e. most difficult) problems. The purpose of this section is to perform a more detailed analysis of all four solution methods on more difficult problem instances. Toward this end, we searched for other factors besides problem size that impact problem difficulty. During this search, we found that when *TotalS* is close to *TotalD*, the problem becomes harder to solve to optimality. Consequently, five additional sets of problem instances with *TotalS* - *TotalD* less than or equal to 10 were created. Ten instances are considered in each category.

Table 6-18 shows the criteria defing the five categories of problem instances considered in this section. In all problem instances *TotalS* – *TotalD* is less than or equal to 10 units.

Table 6-18. Categories of hard problem instances

Instance category	# of demanders & suppliers	Length of cycle (*T*)	*TotalS*- *TotalD*
d06s06t030	6	30	≤ 10
d06s06t100	6	100	≤ 10
d10s10t010	10	10	≤ 10
d10s10t030	10	30	≤ 10
d10s10t100	10	100	≤ 10

Tables 6-19, 6-20, 6-21, and 6-22 respectively show the results where following methods are used to solve the hard problem instances: pure CPLEX, CPLEX initialized with a feasible solution, simulated annealing, and the random algorithm. We can see that most of the objective values are much higher in these tables than the tables in Section 6.4. Consider the results for problem size d06s06t30. Section 6.4 also considered the same problem size d06s06t30 but the average result for objective 1 and objective 2 was much lower in Section 6.4 than in this section. Indeed, the average value of objective 1 for the pure CPLEX algorithm is 3.9 for the easy problem instances (Table 6-12) but is 30.6 for the hard problem instances (Table 6-19). This phenomenon can be explained by the requirement that *TotalS* – *TotalD* ≤*10* for the hard instances. This requirement limits the decision maker's options regarding supplies and supply subtraction epochs.

Table 6-19. Results for CPLEX without an initial feasible solution (hard instances)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d06s06t30	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	373	412	343	243	299	150	275	209	217	303	282.4
	OBJ2	11	9	10	8	9	7	11	7	6	9	8.7
d10s10t10	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
d10s10t30	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	373	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

Table 6-20. Results for CPLEX with an initial feasible solution (hard instances)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d06s06t30	OBJ1	41	26	**45**	23	13	14	30	**48**	**33**	**46**	31.9
	OBJ2	**4**	3	**6**	4	2	3	4	**6**	**6**	**6**	4.4
d06s06t100	OBJ1	389	380	372	238	266	186	253	268	232	250	283.4
	OBJ2	11	11	9	7	9	7	10	7	6	8	8.5
d10s10t10	OBJ1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
	OBJ2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
d10s10t30	OBJ1	34	10	10	2	14	8	27	33	27	1	16.6
	OBJ2	5	3	3	2	4	3	4	3	5	1	3.3
d10s10t100	OBJ1	371	394	331	306	437	264	419	319	258	329	342.8
	OBJ2	11	11	10	9	13	14	11	7	14	7	10.7

Bold: Optimal solution

The results for pure CPLEX (Table 6-19) and CPLEX with an initial feasible solution (Table 6-20) are very similar. Table 6-21 shows the results for the simulated annealing algorithm with *P*=1000 and α=0.999. Table 6-23 shows the overall results for the hard problem instances. These results show many of the same trends that were observed for the easy instances. In particular, the random algorithm is not performing well. Also, the SA algorithm performs better than the random algorithm but usually not as well as the CPLEX-based algorithms.

Table 6-21. Results for the simulated annealing algorithm (hard instances)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d06s06t30	OBJ1	66	79	70	49	58	51	62	81	72	56	64.4
	OBJ2	**4**	3	**6**	4	4	5	5	7	7	7	5.2
d06s06t100	OBJ1	433	513	399	335	253	154	330	306	325	306	335.4
	OBJ2	11	12	11	7	9	7	8	7	7	9	8.8
d10s10t10	OBJ1	2	5	5	**0**	2	**0**	**0**	1	3	2	2
	OBJ2	**0**	1	**0**	**0**	**0**	**0**	**0**	**0**	1	**0**	0.2
d10s10t30	OBJ1	71	29	66	39	33	50	45	66	64	29	49.2
	OBJ2	3	4	4	3	3	3	3	4	5	3	3.5
d10s10t100	OBJ1	341	324	369	349	450	365	296	351	336	352	353.3
	OBJ2	8	8	8	8	9	9	9	6	7	8	8

Bold: Optimal solution

Table 6-22. Results for the random algorithm (hard instances)

Problem size	Objective	Instance										Avg.
		0	1	2	3	4	5	6	7	8	9	
d06s06t30	OBJ1	90	100	97	84	101	96	76	97	108	87	93.6
	OBJ2	8	8	9	8	8	8	8	8	8	8	8.1
d06s06t100	OBJ1	483	723	541	529	579	413	504	470	418	465	512.5
	OBJ2	12	17	12	12	13	10	11	11	9	10	11.7
d10s10t10	OBJ1	10	11	10	10	8	10	10	11	10	8	9.8
	OBJ2	3	3	3	3	2	3	2	3	2	3	2.7
d10s10t30	OBJ1	108	110	104	105	102	125	109	99	121	73	105.6
	OBJ2	9	10	10	9	9	10	10	9	10	6	9.2
d10s10t100	OBJ1	593	674	581	694	710	751	791	479	715	690	667.8
	OBJ2	13	15	14	15	17	18	19	12	16	17	15.6

Interestingly, there is only one combination of hard instance category and objective—d10s10t100 with objective 2—in which the SA algorithm outperforms a CPLEX-based algorithm. This indicates that the SA algorithms advantage over traditional integer programming is mainly limited to the largest problem instances, not the "tight" instances where *TotalS – TotalD ≤10*.

Table 6-23. Overall experimental results (hard instances)

Problem size	Objective	CPLEX w/o Initial Feas. Soln.	CPLEX w/ Initial Feas. Soln.	SA	Random
d06s06t30	Objective 1	**30.6**	31.9	64.4	93.6
	Objective 2	**4.2**	4.4	5.2	8.1
d06s06t100	Objective 1	**282.4**	283.4	335.4	512.5
	Objective 2	8.7	**8.5**	8.8	11.7
d10s10t10	Objective 1	**0**	**0**	2	9.8
	Objective 2	**0**	**0**	0.2	2.7
d10s10t30	Objective 1	17.6	**16.6**	49.2	105.6
	Objective 2	**3**	3.3	3.5	9.2
d10s10t100	Objective 1	345.4	**342.8**	353.3	667.8
	Objective 2	11	10.7	**8**	15.6

Bold: Best performance among the four methods

Figure 6-7 shows the individual impact of the solution method (left) and problem size (right) on the best value that is found for objective 1. This figure shows that the CPLEX-based methods have better results than other methods. In addition, the pure CPLEX algorithm performs slightly better on average than the method where CPLEX is initialized with a feasible solution. Note that the objective value goes up as the number of demanders and suppliers and the cycle length increase.

Figure 6-8 shows the combined impact of the solution method and problem size on the best value that is found for objective 1. This figure indicates that the objective value is higher for problems with a large cycle length than those with a small cycle length.

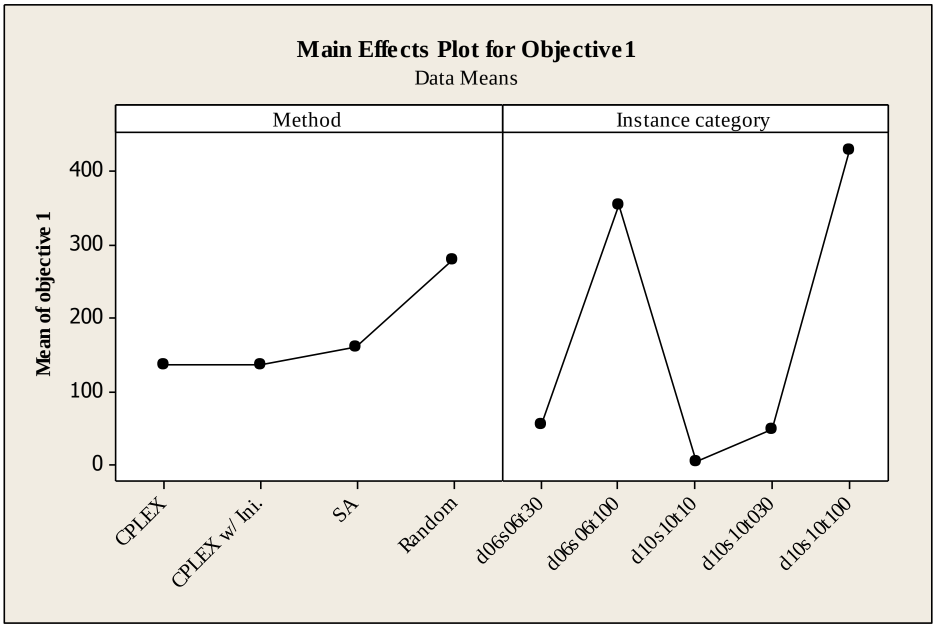

Figure 6-7. Avg. value of objective 1 by method (left) and by problem size (right) (hard instances)

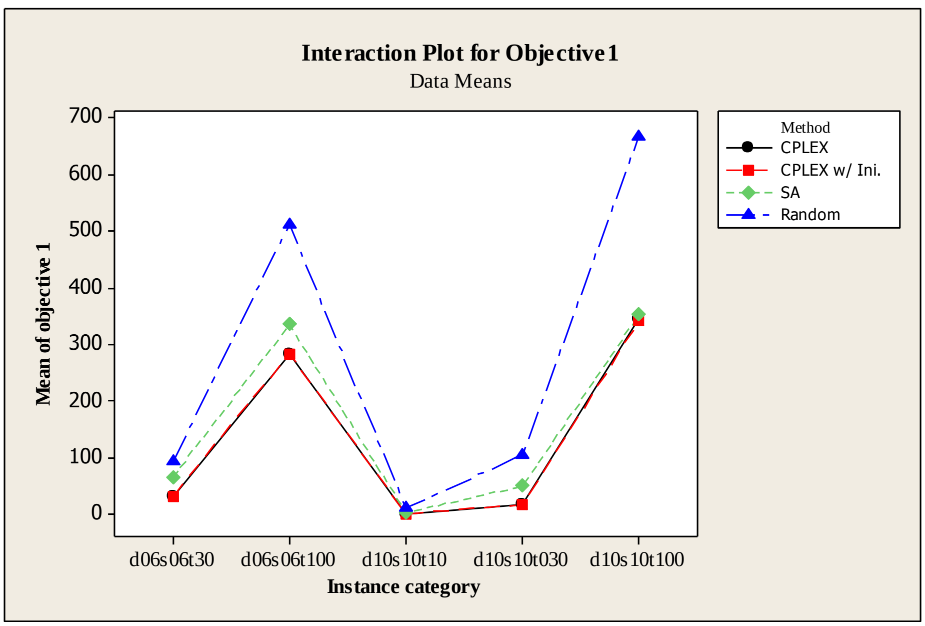

Figure 6-8. Avg. value of objective 1 achieved for each combination of method and problem size (hard instances)

Figure 6-9 shows the individual impact of the solution method (left) and problem size (right) on the best value that is found for objective 2. This figure shows that the SA algorithm's overall performance is slightly better than the other methods for objective 2. This is due to its much better performance than the other methods for instance category d10s10t100. We also observe that the objective value goes up as the number of demanders and suppliers and the cycle length increase.

Figure 6-10 shows the combined impact of the solution method and problem size on the best value that is found for objective 2. This figure indicates that the objective value is higher for problems with a large cycle length than those with a small cycle length.

Overall, the results from Sections 6.4 and 6.5 indicate that traditional integer programming using the CPLEX solver is a good method for solving this problem. Simulated annealing can be also useful for solving the largest problems.

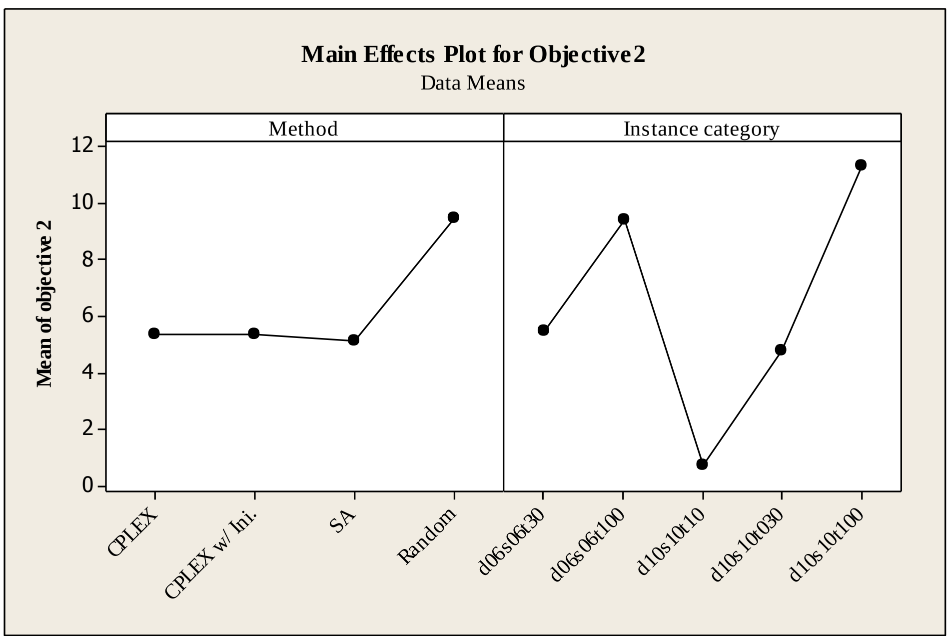

Figure 6-9. Avg. value of objective 2 by method (left) and by problem size (right) (hard instances)

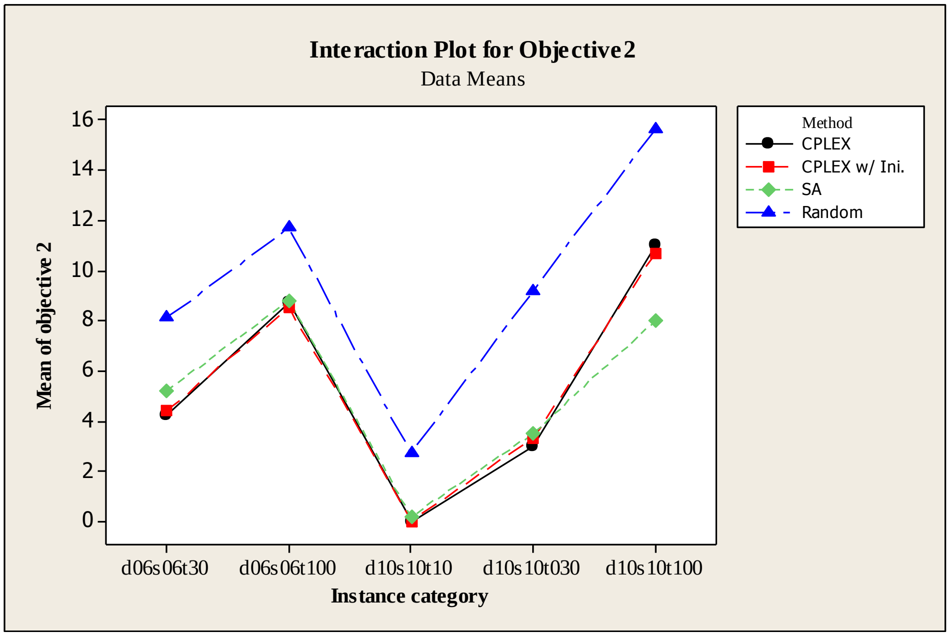

Figure 6-10. Avg. value of objective 2 achieved for each combination of method and problem size (hard instances)

CHAPTER 7: CONCLUSIONS AND FUTURE RESEARCH

This thesis introduces a new optimization problem to the operations research literature: optimal cyclic control of buffer between two non-synchronized manufacturing processes. This problem is formally defined and modeled as an integer linear program (ILP). Two theorems concerning (1) problem feasibility and (2) tightening the ILP are proved. Four solution methods are proposed for solving this problem: pure integer programming using CPLEX, CPLEX initialized with a feasible solution, simulated annealing, and a random algorithm. These methods are compared in two sets of experiments.

Results show that traditional integer programming is a good method for attacking this problem. However, even this method begins to show limitations when facing a large problem or a problem where the total supply quantity is close to the total demand quantity. For the largest problems, simulated annealing exhibits better performance than other methods.

In the future, there are some more aspects that we can consider. This thesis can be extended in several directions. First, we can consider both objectives simultaneously. Second, we could incorporate delivery distances and delays into the problem. Finally, the objective could consider not only the buffer inventory, but also the cost for each supplier to replenish the buffer. These costs could be different for different suppliers.

REFERENCES

Abuhilal, L., Rabadi, G., & Sousa-Poza, A. (2006). Supply chain inventory control: A comparison among JIT, MRP, and MRP with information sharing using simulation. *Engineering Management Journal, 18*(2), 51-57.

Alfieri, A., & Matta, A. (2012). Mathematical programming formulations for approximate simulation of multistage production systems. *European Journal of Operational Research, 219*(3), 773-783.

Ardalan, A. (1997). Analysis of Local Decision Rules in a Dual-Kanban Flow Shop. *Decision Sciences, 28*(1), 195-211.

Buzacott, J., & Shanthikumar, J. (1994). Safety stock versus safety time in MRP controlled production systems. *Management Science, 40*(12), 1678-1689.

Chu, C.-H., & Shih, W.-L. (1992). Simulation studies in JIT production. *The International Journal Of Production Research, 30*(11), 2573-2586.

Chuah, K. H. (2004). *Optimization and simulation of just-in-time supply pickup and delivery systems*. University of kentucky.

Deleersnyder, J.-L., Hodgson, T. J., Muller-Malek, H., & O'Grady, P. J. (1989). Kanban controlled pull systems: an analytic approach. *Management Science, 35*(9), 1079-1091.

Dobson, G., & Yano, C. A. (1994). Cyclic scheduling to minimize inventory in a batch flow line. *European Journal of Operational Research, 75*(2), 441-461.

Dong, Y., Carter, C. R., & Dresner, M. E. (2001). JIT purchasing and performance: an exploratory analysis of buyer and supplier perspectives. *Journal of Operations Management, 19*(4), 471-483.

Fernandez, M., Li, L., & Sun, Z. (2013). “Just-for-Peak” buffer inventory for peak electricity demand reduction of manufacturing systems. *International Journal of Production Economics, 146*(1), 178-184.

Florian, M., Lenstra, J. K., & Rinnooy Kan, A. (1980). Deterministic production planning: Algorithms and complexity. *Management Science, 26*(7), 669-679.

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

Halim, A. H., & Ohta, H. (1994). Batch-scheduling problems to minimize inventory cost in the shop with both receiving and delivery just-in-times. *International Journal of Production Economics, 33*(1), 185-194.

Harris, F. W. (1990). How many parts to make at once. *Operations Research, 38*(6), 947-950.

Hay, E. J. (1988). *The just-in-time breakthrough: implementing the new manufacturing basics*: Wiley.

Iwase, M., & Ohno, K. (2011). The performance evaluation of a multi-stage JIT production system with stochastic demand and production capacities. *European Journal of Operational Research, 214*(2), 216-222.

Khan, L. R., & Sarker, R. A. (2002). An optimal batch size for a JIT manufacturing system. *Computers & Industrial Engineering, 42*(2), 127-136.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671-680.

Kneppelt, L. R. (1984). Product structuring considerations for master production scheduling. *Prod. Inventory Manage, 25*(1), 83-99.

Lee, H.-G., 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.

Mascolo, M. D., Frein, Y., & Dallery, Y. (1996). An analytical method for performance evaluation of kanban controlled production systems. *Operations Research, 44*(1), 50-64.

Matta, A., Dallery, Y., & Di Mascolo, M. (2005). Analysis of assembly systems controlled with kanbans. *European Journal of Operational Research, 166*(2), 310-336.

Mauro, J. J. P. (2008). *Strategic inventory management in an aerospace supply chain.* Massachusetts Institute of Technology.

McDonald, C. M., & Karimi, I. A. (1997). Planning and scheduling of parallel semicontinuous processes. 1. Production planning. *Industrial & Engineering Chemistry Research, 36*(7), 2691-2700.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics, 21*(6), 1087-1092.

Metters, R. (1997). Quantifying the bullwhip effect in supply chains. *Journal of Operations Management, 15*(2), 89-100.

Mitra, D., & Mitrani, I. (1990). Analysis of a kanban discipline for cell coordination in production lines. I. *Management Science, 36*(12), 1548-1566.

Mitra, D., & Mitrani, I. (1991). Analysis of a kanban discipline for cell coordination in production lines, II: Stochastic demands. *Operations Research, 39*(5), 807-823.

Newman, W. R., Hanna, M., & Maffei, M. J. (1993). Dealing with the uncertainties of manufacturing: flexibility, buffers and integration. *International Journal of Operations and Production Management, 13*, 19-19.

Radhoui, M., Rezg, N., & Chelbi, A. (2009). Integrated model of preventive maintenance, quality control and buffer sizing for unreliable and imperfect production systems. *International Journal of Production Research, 47*(2), 389-402.

Roy, M. D., Sana, S. S., & Chaudhuri, K. (2012). An integrated producer–buyer relationship in the environment of EMQ and JIT production systems. *International Journal of Production Research, 50*(19), 5597-5614.

Salameh, M., & Ghattas, R. (2001). Optimal just-in-time buffer inventory for regular preventive maintenance. *International Journal of Production Economics, 74*(1), 157-161.

Sarker, B. R., & Parija, G. R. (1994). An optimal batch size for a production system operating under a fixed-quantity, periodic delivery policy. *Journal of the Operational Research Society*, 891-900.

Tang, O., & Grubbström, R. W. (2002). Planning and replanning the master production schedule under demand uncertainty. *International Journal of Production Economics, 78*(3), 323-334.

Wang, H., & Wang, H.-P. (1990). Determining the number of kanbans: A step toward non-stock-production. *The International Journal Of Production Research, 28*(11), 2101-2115.

Whybark, D. C., & Williams, J. G. (1976). Material requirements planning under uncertainty. *Decision Sciences, 7*(4), 595-606.

Xu, Z. (2004). *Two approaches to buffer management under demand uncertainty: an analytical process.* Massachusetts Institute of Technology.