International Journal of Industrial Engineering, 15(3), 245-256, 2008

Offsetting Inventory Cycles using Mixed Integer Programming and Genetic Algorithm

I.K. Moon[1], B.C. Cha[2], and S.K. Kim[3]

[1] Department of Industrial Engineering,
Pusan National University, Busan, 609-735, Korea

[2] Postal Technology Research Center,
ETRI, Daejeon, 305-700, Korea

[3] S&T Daewoo Co., Ltd,
Busan, 619-873, Korea

Corresponding author's e-mail: {IK Moon, ikmoon@pusan.ac.kr}

We propose a mixed integer programming model to minimize the maximum storage space requirement over an infinite time horizon by offsetting the inventory cycles of items. We also develop a genetic algorithm to find the near-optimal solution. The mixed integer programming model and the genetic algorithm produce better results than the existing heuristic. We also develop a mixed integer programming model for the finite time horizon; this model is more general and realistic than that for the infinite time horizon. A warehouse management system is designed based on the algorithms we developed.

Significance: We provide both a mixed integer program and a genetic algorithm to minimize the maximum storage space requirement for the warehouse. These methods can be practically implemented at warehouses and distribution centers for the utilization of the storage space and other resources.

Keywords: Offsetting cycles, Inventory, Mixed integer programming model, Genetic algorithm.

(Received 15 December 2007; Accepted in revised form 20 July 2008)

1. INTRODUCTION

In today's highly competitive global market, businesses are seeking ways to reduce cost without compromising on product quality or customer service. In recent years, various innovative manufacturing philosophies have been adopted by manufacturing firms in order to improve competitiveness (Banerjee and Banerjee, 1994). Reducing inventory storage space and other relevant costs is one major focus for many of these philosophies, for instance, the just-in-time philosophy.

In inventory systems cost savings can be achieved by coordinating the replenishment of several items. The joint replenishment problem (JRP) deals with the problem of coordinating the replenishment of a group of items that may be jointly ordered from a single supplier. In such systems, the ordering cost has two components: a major common ordering cost incurred whenever an order is placed and a minor ordering cost incurred if an item is ordered. During the last three decades, the JRP has received considerable attention from researchers. Arkin et al. (1989) proved that the JRP is an NP-hard problem; therefore, it is unlikely that the JRP is solvable by polynomial-time algorithms. Silver (1975, 1976) discussed the advantages and disadvantages of coordinating replenishments and presented a very simple non-iterative procedure to solve it. Kaspi and Rosenblatt (1983) proposed an approach based on attempting several values of the basic cycle time between the minimum and the maximum values. Then, they employed the heuristic of Kaspi and Rosenblatt (1991) for each value of the basic cycle time, which is a modified version of the algorithm of Silver (1975). They demonstrated that their procedure (known as the RAND algorithm) outperforms all the available heuristics. Later, Goyal and Deshmukh (1993) proposed an improvement of the lower bound used by Kaspi and Rosenblatt (1991). Wildeman et al. (1997) presented a new solution approach based on Lipschitz optimization to obtain a solution with an arbitrarily small deviation from the optimal. Moon et al. (2008) extended the JPR to multiple suppliers case in which each supplier has different discount scheme. Recently, Khouja and Goyal (2008) reviewed and summarized the literature on the JRP since 1989.The coordinating replenishment is also important in one-warehouse multiple retailer system. Refer Hong and Park (2006), Monthatipkul and Yenradee (2007) and Cha et al. (2008) for the most recent works on this topic. Kim et al. (2006) developed a mathematical model to decide a joint production-shipment policy in a supply chain with a single manufacturer and multiple retailers.

There are many resource restrictions in actual production/inventory systems (for example, budget, storage, and transportation capacity). Goyal (1975) introduced a JRP with one resource constraint and developed a heuristic algorithm using the Lagrangian multiplier. Moon and Cha (2006) developed both a modified RAND algorithm and a genetic algorithm for the JRP with resource restrictions. Hoque (2006) developed a generalized global optimal algorithm for the JRP with storage and transport capacities and budget constraints. But in the general JRP, since every item is replenished simultaneously at the beginning of the time horizon, the storage space is needed maximumly at first and used inefficiently at the rest of the time. We can consider offsetting inventory cycles of items to make up this weak point of the JRP. Murthy et al. (2003) formulated the problem and developed a scheme to offset the inventory cycles of items on the time axis in a manner that minimizes the maximum storage space requirement over an infinite time horizon. Their study provided insights into finding a solution to the problem.

In this paper, we propose a mixed integer programming model to minimize the maximum storage space requirement over an infinite time horizon by offsetting the inventory cycles of items. Since the mixed integer programming models cannot be used for solving large size problems, we also develop a complete procedure of the genetic algorithm to find the near-optimal solution for the model. In addition, we consider a more realistic situation of the time horizon. We extend the mixed integer programming model to the finite time horizon cases in which there exists a finite horizon for each item or a common finite horizon for all the items. We show that the finite time horizon model is more reasonable in using offsetting inventory cycles of items sharing storage than the infinite time horizon model.

The rest of this paper is organized as follows. In Section 2, we present the problem statement, assumptions, and notation and formulate a mixed integer programming model. We develop a genetic algorithm in Section 3. In Section 4, we present numerical examples and compare them with the algorithm of Murthy et al. (2003). We develop a mixed integer programming model with a finite time horizon in Section 5. We present numerical examples and compare them with the model without offsetting. Further, we show the performance of the genetic algorithm using randomly generated problems. A warehouse management system is designed based on the algorithms. Finally, Section 7 presents the conclusions.

2. MIXED INTEGER PROGRAMMING MODEL FOR THE INFINITE TIME HORIZON

As shown in the following example of Murthy et al. (2003), the offsetting of cycles does not alter the trade-off between the ordering and holding costs and reduces the maximum storage space requirement. Note that the maximum storage space requirement is 2Q for the case without offsetting and it has been reduced to 1.75Q for the case with offsetting.

Table 1: An example problem

Item	Storage space per unit item	Time between orders	Order quantity
#1	s (=1)	$4T$	Q
#2	s (=1)	$2T$	Q

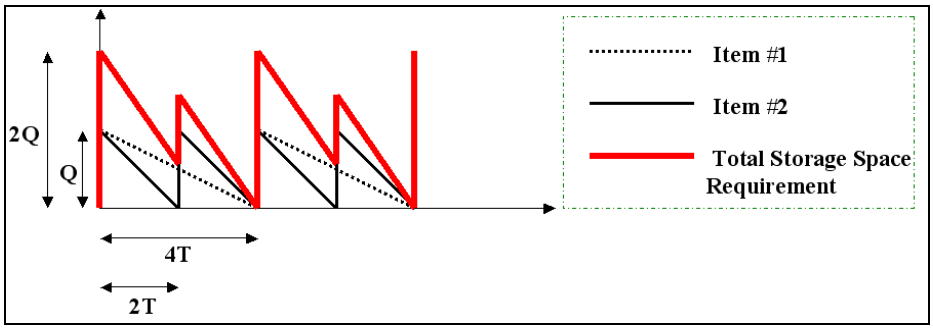

Figure 1. Example problem without cycle offsetting.

The following assumptions and notations, which are the same as those in Murthy et al. (2003), are used in developing the mixed integer programming model:

- The demand rate is deterministic and constant.
- The replenishment is instantaneous.
- The time between orders is known and constant for all items over an infinite time horizon.
- The time between orders for all items is restricted to be integer multiples of a basic period.

- The lead time is known and constant for all items. Further, we can set the lead time as zero for all the items without loss of generality of the solution procedure.
- Shortages and backlogs are not allowed.

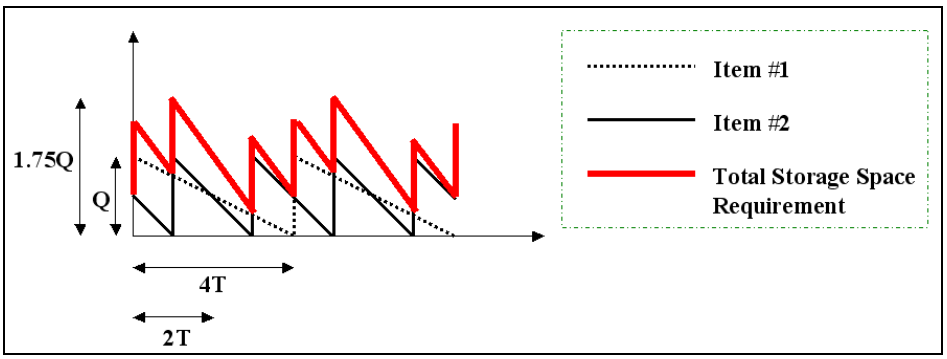

Figure 2. Example problem with cycle offsetting.

LCM: lowest common multiple of the replenishment cycles
S: maximum storage space required for all items in the time interval t=0 to t=LCM-1
s_i: storage space required per unit of item i (set to 1 for all items)
I_{it}: inventory level for item i at the beginning of period t (decision variable)
X_{it}: binary variable indicating whether an order for item i occurs at the beginning of period t (decision variable)
n: number of items
D_i: demand per basic period for item i
TBO_i: time between orders for item i
Q_i: replenishment quantity for item i, $Q_i = D_i \times \text{TBO}_i$

The TBO for each item is an integer multiple of the basic period. The basic period and TBO of each item are obtained from the solution approaches of the general JRP. The objective function (2.1) is to minimize the maximum storage space requirement over an infinite time horizon. Since the total storage space requirement pattern is periodic, the maximum must occur in the time interval of t=0 to t=LCM(TBO_1,..., TBO_n)−1. Although the time horizon is infinite, we need not consider the time after the LCM due to the periodic nature of the inventory cycles.

Minimize S (2.1)

subject to

$$\sum_{t=0}^{\text{TBO}_i-1} X_{it} = 1, \qquad i = 1, 2, \cdots, n \tag{2.2}$$

$$I_{i0} = Q_i X_{i0} + \frac{Q_i}{\text{TBO}_i} \sum_{t=1}^{\text{TBO}_i-1} t X_{it}, \quad i = 1, 2, \cdots, n \tag{2.3}$$

$$I_{it} = I_{i(t-1)} + Q_i X_{it} - D_i, \qquad i = 1, 2, \cdots, n \;\; t = 1, \cdots, \text{TBO}_i\text{-}1 \tag{2.4}$$

$$S \geq \sum_{i=1}^{n} I_{ik(i)}, \qquad k(i) = \begin{cases} t, & \text{if } t < \text{TBO}_i \\ \text{Mod}(t / \text{TBO}_i), & \text{if } t \geq \text{TBO}_i \end{cases} \tag{2.5}$$

$$t = 0, 1, \cdots, \text{LCM}(\text{TBO}_1, \cdots, \text{TBO}_n) - 1$$

$$X_{it} = 0 \text{ or } 1$$

Constraint (2.2) determines the earliest replenishment time period for each item. X_{it}=1 indicates that item i is replenished at time t (from t=0 to TBO_i−1). Constraint (2.3) represents the inventory level for each item at t=0 as a function of X_{it}. Constraint (2.4) shows the general inventory balance equations. Constraint (2.5) maintains the periodic nature of the inventory cycles at LCM-1.

3. GENETIC ALGORITHM

In this section, the main ideas of the genetic algorithm are introduced and we demonstrate how the genetic algorithm can be applied to our problem. Genetic algorithms, which have been widely used to solve operations management problems during the last decade, are stochastic search algorithms based on the mechanism of natural selection and natural genetics. Unlike conventional search techniques, genetic algorithms begin with an initial set of (random) solutions known as a population. Each individual in the population is known as a chromosome, that represents a solution to the problem at hand. The chromosomes evolve through successive iterations, that are called generations. During each generation, the chromosomes are evaluated using some measures of fitness. In general, the genetic algorithm is applied to spaces that are too large to be exhaustively searched. It is commonly accepted that any genetic algorithm that is used to solve a problem must have basic components; however, it can have different characteristics depending on the problem under study.

We explain our overall strategies including the chromosome style under the following headings:

- Representation and initialization
- Objective and fitness function
- Reproduction, crossover, and mutation

3.1 Representation and initialization

The appropriate representation of a solution plays a key role in the development of a genetic algorithm. In this paper, the length of the chromosome is the total number of items and each gene in the chromosome represents the earliest replenishment time period of each item. This implies that gene i must have an integer value between 0 and TBO_i-1. In our study, the initial population of chromosomes is generated randomly. We generate a random number between 0 and 1 and convert it such that it takes a feasible integer value between 0 and TBO_i-1 for each gene of all the chromosomes. Each chromosome in the population represents a potential solution to the problem. All the genes are decoded to evaluate each chromosome. For the example provided in Murthy et al. (2003), both the representation and decoding process of a chromosome are shown in Figure 3.

Item i	1	2	3	4	5	6	7	8	9
TBO_i	4	5	9	12	15	8	6	12	2

(0,1) RN	0.0138	0.1121	0.9841	0.6448	0.8534	0.7991	0.7176	0.1167	0.5637
		$\lfloor TBO_i \times Gene(i) \rfloor$							
Chromosome	**0**	**0**	**8**	**7**	**12**	**6**	**4**	**1**	**1**
		$X_{it}=1$ *and* $X_{it'}=0$ *for* $t \neq t'$							
Decoding	$X_{1,0}=1$	$X_{2,0}=1$	$X_{3,8}=1$	$X_{4,7}=1$	$X_{5,12}=1$	$X_{6,6}=1$	$X_{7,4}=1$	$X_{8,1}=1$	$X_{9,1}=1$

Figure 3: Chromosome and decoding process.

In Figure 3, the randomly generated real number of the first gene in a chromosome is converted to the feasible integer value that indicates the earliest replenishment time period of the first item.

$$\lfloor TBO_1 \times Gene(1) \rfloor = \lfloor 4 \times 0.0138 \rfloor = 0$$

For the evaluation of a chromosome, this integer value is decoded to determine the value of the decision variable X_{1t} ($X_{1,0}=1$, $X_{1,1}=0$, $X_{1,2}=0$, $X_{1,3}=0$).

3.2 Objective and fitness function

As mentioned above, each chromosome in the population is evaluated by the mixed integer programming model. With the decoded values of each chromosome, which are presented in Figure 3, the objective value S of a fitness function can be obtained from equations (2.3), (2.4) and (2.5). This pertains to the determination of the maximum storage space requirement in the time interval of t=0 to t=LCM-1 with the earliest replenishment time period for each item.

$$\text{Fitness(chromosome)}\ S = \max\left\{\sum_{i=1}^{n} I_{i0}, \sum_{i=1}^{n} I_{i1}, \cdots, \sum_{i=1}^{n} I_{i(LCM-1)}\right\}$$

3.3 Reproduction, crossover, and mutation

Various evolutionary methods can be applied to obtain a good solution to this problem. We use ranking selection to select individuals for reproduction. We produce offsprings through a uniform crossover with probability P_c. Whenever an offspring is produced, mutation is applied with probability P_m. The operation of mutation replaces one gene of the chromosome chosen at random with a new random number between 0 and 1. In order to choose appropriate parameter values in our genetic algorithms for the offsetting inventory cycles with the objective of minimizing the maximum storage space, we employed the abovementioned crossover operator and mutation operator with the following parameters through a pilot test:

- Population size: $N_{pop} = 50$
- Crossover probability: $P_c = 0.5$
- Mutation probability: $P_m = 0.05$

The termination condition was to stop the algorithm when the best individual does not improve over 500 generations. Let P(t) and C(t) be the respective populations of parents and offsprings in generation t. The overall procedure of the proposed genetic algorithm is described in Figure 4.

```
begin
  Initialize P(t);
    t ← 0;
    set N_pop, P_c and P_m;
    generate real numbers of n× N_pop between 0 to 1 randomly.
  Evaluate P(t);
    convert and decode P(t) to obtain the values of decision variable X_it;
    evaluate P(t) by equations of the mixed integer programming model;
    update the best solution;
  while (no termination condition) do
    begin
      t ← t+1;
      Select P(t) from P(t-1) by ranking selection rule;
      Generate C(t) from P(t) by applying the crossover and mutation operations;
      Alter P(t) ← P(t) ∪ C(t);
      Evaluate P(t);
        convert and decode P(t) to obtain the values of decision variable X_it;
        evaluate P(t) by equations of the mixed integer programming model;
        update the best solution;
    end
end
```

Figure 4: The procedure of the proposed genetic algorithm.

4. NUMERICAL EXAMPLES

Our problem is formulated by the mixed integer program developed in Section 2 and solved with LINDO software program on a Pentium PC. The genetic algorithm was coded using EVOLVER 4.0 software program (http://www.hearne.com.au/products/evolver/) with Microsoft Excel and Visual Basic interface on the same PC. This facilitated easy data input. We will explain the meaning of the solution by using an example. We use the same example used in Murthy et al. (2003), and the data are shown in Table 2.

Table 3 shows the optimal solution of the mixed integer programming model and compares it with the solution in Murthy et al. (2003). Note that the genetic algorithm also leads to the optimal solution. The maximum storage space obtained by Murthy et al. (2003) has been reduced by 10.17%. The maximum storage space obtained without offsetting has been reduced by 24.06%. In order to compare the solutions, we illustrate the change of the total space requirement over time in Figures 5, 6, and 7.

Table 2: Example problem

Item i	1	2	3	4	5	6	7	8	9
Q_i	100	200	81	144	150	160	90	60	50
TBO_i	4	5	9	12	15	8	6	12	2

Table 3: Comparison of solutions

	Without Offsetting		Murthy et al.'s Heuristic		MIP (Optimal)		Genetic Algorithm	
Item	X_{it}=1	I_{i0}	X_{it}=1	I_{i0}	X_{it}=1	I_{i0}	X_{it}=1	I_{i0}
1	$X_{1,0}$	100	$X_{1,0}$	100	$X_{1,0}$	100	$X_{1,0}$	100
2	$X_{2,0}$	200	$X_{2,4}$	160	$X_{2,0}$	200	$X_{2,0}$	200
3	$X_{3,0}$	81	$X_{3,0}$	81	$X_{3,8}$	72	$X_{3,8}$	72
4	$X_{4,0}$	144	$X_{4,0}$	144	$X_{4,7}$	84	$X_{4,7}$	84
5	$X_{5,0}$	150	$X_{5,0}$	150	$X_{5,12}$	120	$X_{5,12}$	120
6	$X_{6,0}$	160	$X_{6,6}$	120	$X_{6,6}$	120	$X_{6,6}$	120
7	$X_{7,0}$	90	$X_{7,5}$	75	$X_{7,4}$	60	$X_{7,4}$	60
8	$X_{8,0}$	60	$X_{8,4}$	20	$X_{8,11}$	5	$X_{8,11}$	5
9	$X_{9,0}$	50	$X_{9,1}$	25	$X_{9,1}$	25	$X_{9,1}$	25
Sum		1035		875		786		786
Reduction		-		15.46%		24.06%		24.06%

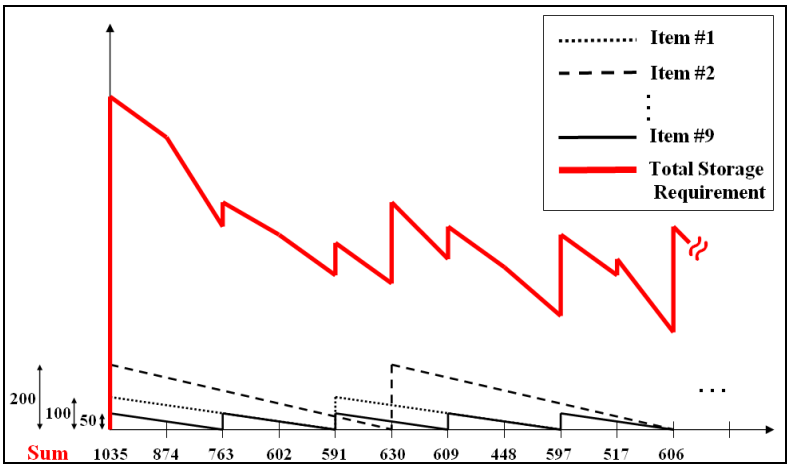

Figure 5. Case without offsetting.

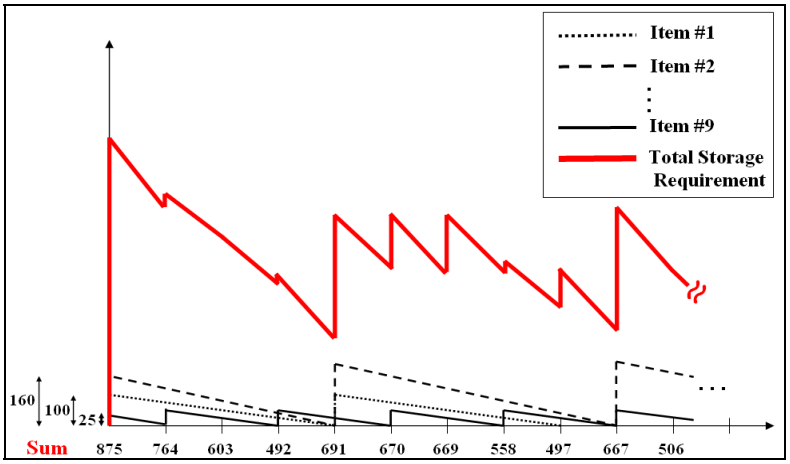

Figure 6. Heuristic algorithm.

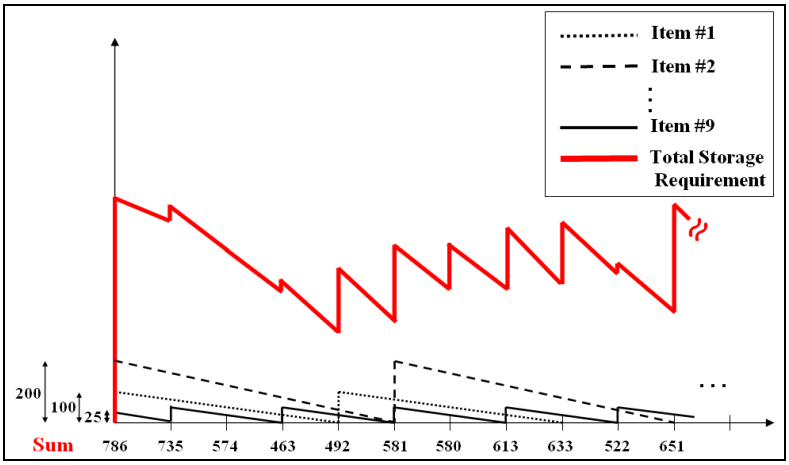

Figure 7. MIP & GA.

5. FINITE TIME HORIZON MODEL

In this section, we consider a more realistic situation of the time horizon. In practical terms, if the number of items increases, then the LCM of the replenishment cycles will increase exponentially. Suppose that the basic period T is one day. An LCM of 1,000 indicates that the length of the time horizon is approximately 3 years; that is somewhat unrealistic. The assumption of an infinite time horizon of all items is impractical. Due to this, we develop the mixed integer programming model with a finite time horizon. Note that the algorithm in Murthy et al. (2003) can be applied only to the infinite case. We consider a finite time horizon in this section. There are two possible cases. First, all the items may have a common finite horizon. Second, each item may have a different finite horizon.

The assumptions are the same as those in Section 2, except for the assumption of the infinite time horizon that has been changed to the finite time horizon of all items. We require two additional parameters:

F_i: finite time horizon of item i
$F_{\max}$: common finite time horizon of all items, $F_{\max} \le \max\{F_1, \cdots, F_n\}$

The objective (5.1) is to minimize the maximum storage space requirement over a finite time horizon. If $F_{\max}$ is not greater than the LCM, the maximum space must occur in the time interval of t=0 to t=$F_{\max}$.

Minimize S (5.1)

subject to

$$\sum_{t=0}^{\mathrm{TBO}_i-1} X_{it} = 1, \qquad i = 1, 2, \cdots, n \tag{5.2}$$

$$I_{i0} = Q_i X_{i0} + \frac{Q_i}{\mathrm{TBO}_i} \sum_{t=1}^{\mathrm{TBO}_i-1} t X_{it}, \quad i = 1, 2, \cdots, n \tag{5.3}$$

$$I_{it} = I_{i(t-1)} + Q_i X_{it} - D_i, \qquad i = 1, 2, \cdots, n \;\; t = 1, \cdots, \mathrm{TBO}_i - 1 \tag{5.4}$$

$$S \ge \sum_{\{i:F_i > t\}}^{n} I_{ik(i)}, \qquad k(i) = \begin{cases} t & , \text{if } t < \mathrm{TBO}_i \\ \mathrm{Mod}(t/\mathrm{TBO}_i) & , \text{if } t \ge \mathrm{TBO}_i \end{cases} \tag{5.5}$$

$$t = 0, 1, \cdots, F_{\max}$$

$$X_{it} = 0 \text{ or } 1$$

All the constraints are the same as those in Section 2, except constraint (5.5), which limits the periodic nature of the inventory cycles to $F_{\max}$.

We use the same example as that in Murthy et al. (2003) to compare the finite time horizon case with the infinite time horizon case. In this example, we assume that the common limit of the finite time horizon of all items is 52 time units. The optimal solution of the finite time horizon model is shown and compared in Table 4.

As can be inferred, the finite case requires less storage space since most of the peak storage requirement may occur after $F_{\max}$.

$$S\_{\text{infinite}} = \max\left\{\sum_{i=1}^{n} I_{i0}, \cdots, \sum_{i=1}^{n} I_{iF_{\max}}, \cdots, \sum_{i=1}^{n} I_{i(LCM-1)}\right\} \geq S\_{\text{finite}} = \max\left\{\sum_{i=1}^{n} I_{i0}, \cdots, \sum_{i=1}^{n} I_{iF_{\max}}\right\}$$

The maximum storage space obtained by infinite time horizon has been reduced by 11.20%. The result clearly shows that the finite time horizon model is reasonable in using offsetting inventory cycles of items sharing storage. Our genetic algorithm also obtains the optimal solution for the example in Murthy et al. (2003). Moreover, it takes only a few seconds to obtain the solution by using the genetic algorithm.

Table 4: Comparison of solutions

	Without Offsetting		MIP (infinite case)		MIP (finite case)		GA (finite case)	
Item	X_{it}=1	I_{i0}	X_{it}=1	I_{i0}	X_{it}=1	$I_{i,44*}$	X_{it}=1	$I_{i,44*}$
1	$X_{1,0}$	100	$X_{1,0}$	100	$X_{1,1}$	25	$X_{1,1}$	25
2	$X_{2,0}$	200	$X_{2,0}$	200	$X_{2,4}$	200	$X_{2,4}$	200
3	$X_{3,0}$	81	$X_{3,8}$	72	$X_{3,14}$	18	$X_{3,14}$	18
4	$X_{4,0}$	144	$X_{4,7}$	84	$X_{4,6}$	120	$X_{4,6}$	120
5	$X_{5,0}$	150	$X_{5,12}$	120	$X_{5,11}$	120	$X_{5,11}$	120
6	$X_{6,0}$	160	$X_{6,6}$	120	$X_{6,7}$	60	$X_{6,7}$	60
7	$X_{7,0}$	90	$X_{7,4}$	60	$X_{7,2}$	90	$X_{7,2}$	90
8	$X_{8,0}$	60	$X_{8,11}$	5	$X_{8,1}$	15	$X_{8,1}$	15
9	$X_{9,0}$	50	$X_{9,1}$	25	$X_{9,0}$	50	$X_{9,0}$	50
Sum		1035		786		698		698
Reduction		–		24.06%		32.56%		32.56%

* Peak storage occurs at time period 44.

In order to test the performance of our genetic algorithm for the finite time horizon problems, we have performed two kinds of computational experiments. In the first experiment, we use data similar to that in Murthy et al. (2003), which has been generated randomly from uniform distribution on the TBO interval [2, 15] and the demand rate [5, 40]. We set the finite time horizon of all the items ($F_{\max}$) as follows: Only an LCM that is less than 365 is used; otherwise, $F_{\max}$ is set to 365. We experiment with 50 problems in which the number of items is 10. The computational results are summarized in Table 5.

Table 5: Performance of the genetic algorithm

Number of optimal	Average percentage error	Worst percentage error
14	0.63%	0.92%

Average percentage error: $\text{Average}\left(\frac{GA - Optimal}{GA} \times 100\right)$

Because of their nonlinearity and complexity, offsetting the inventory cycles of items is a typical NP-hard problem. Hence, the mixed integer program applies only to small size problems. In the second experiment, we tested the genetic algorithm on three sets of problems (10 problems in each set). The data sets were generated randomly from uniform distributions on the given intervals and parameters (See Table 6).

Table 6: Randomly generated test problems

Parameters	Set 1	Set 2	Set 3
TBO	[2, 15]	[2, 15]	[2, 15]
Demand rate	[5, 40]	[5, 40]	[5, 40]
$F_{\max}$ of all items	52	104	365

number of items	15, 20, 25, 30	15, 20, 25, 30	15, 20, 25, 30

We report the average percentage reduction and the worst percentage reduction, which are compared with the cases without offsetting (See Table 7). The percentage reduction in peak storage requirement as a result of offsetting for the finite time horizon is 34.43% on an average.

Table 7: Percentage reduction of the genetic algorithm

	Set 1		Set 2		Set 3	
Item	Average	Worst	Average	Worst	Average	Worst
15	34.84%	23.06%	29.30%	22.40%	23.86%	14.02%
20	37.22%	29.44%	31.61%	23.72%	27.47%	18.48%
25	41.45%	32.89%	32.93%	27.19%	29.50%	18.54%
30	46.71%	36.58%	40.47%	30.47%	37.82%	27.30%

$$\text{Average percentage reduction: Average}\left(\frac{\text{Without Offsetting - GA}}{\text{Without Offsetting}}\times 100\right)$$

6. DESIGN OF WAREHOUSE PLANNING SYSTEM

A warehouse management system is designed based on the algorithms we developed. The system is coded using Microsoft .NET Framework. Figure 8 shows the input screen of the system. After we input the basic data such as demand per period, replenishment quantities, replenishment intervals, we can choose among four different algorithms (algorithm without offsetting, Murthy et al.'s heuristics algorithm, MIP, and genetic algorithm). Figure 9 shows the analysis screen which tells the total inventory level in each period which can be easily compared with the warehouse capacity. This system can be used in practice to effectively manage the warehouse. If the total inventory level in any period exceeds the warehouse capacity, the system gives a warning signal so that the manager needs to resolve the problem by changing the replenishment quantities or replenishment intervals.

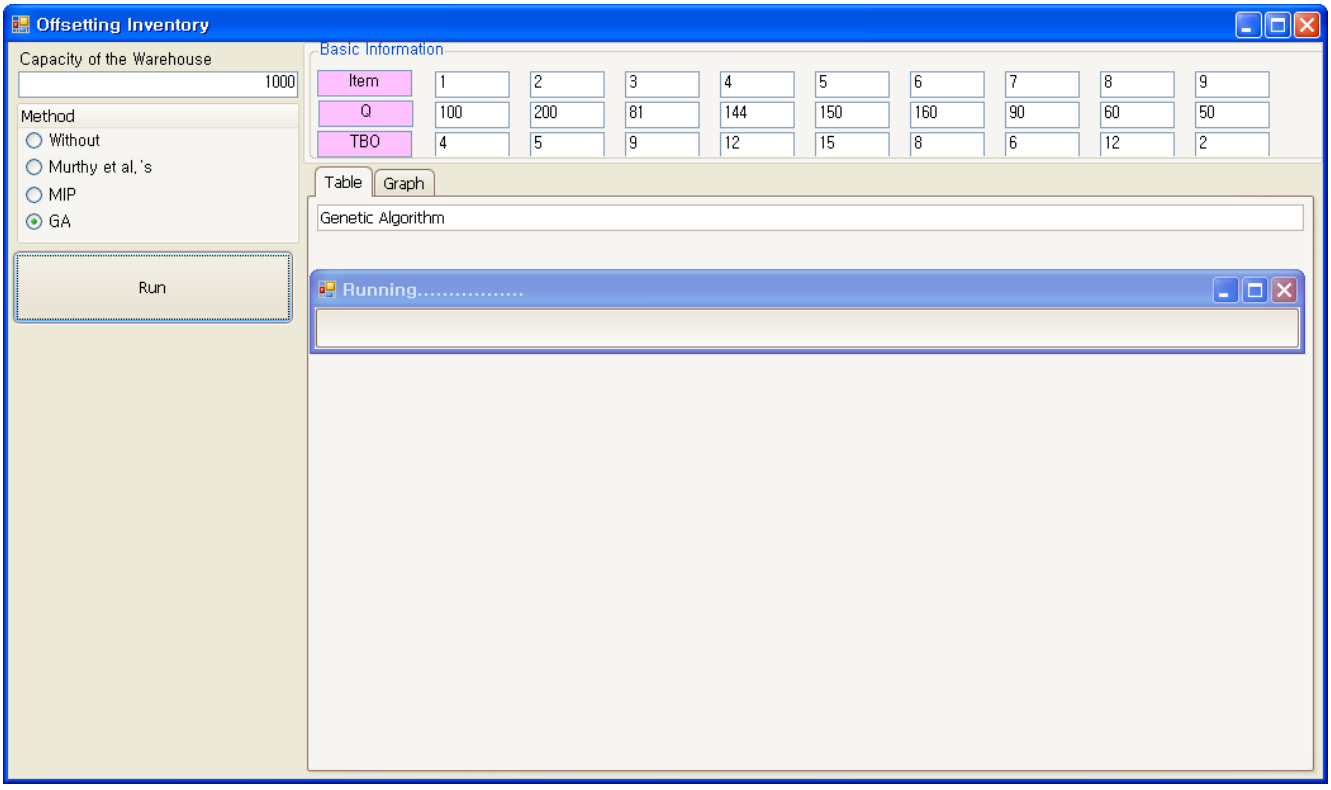

Figure 8: Input screen of a warehouse management system

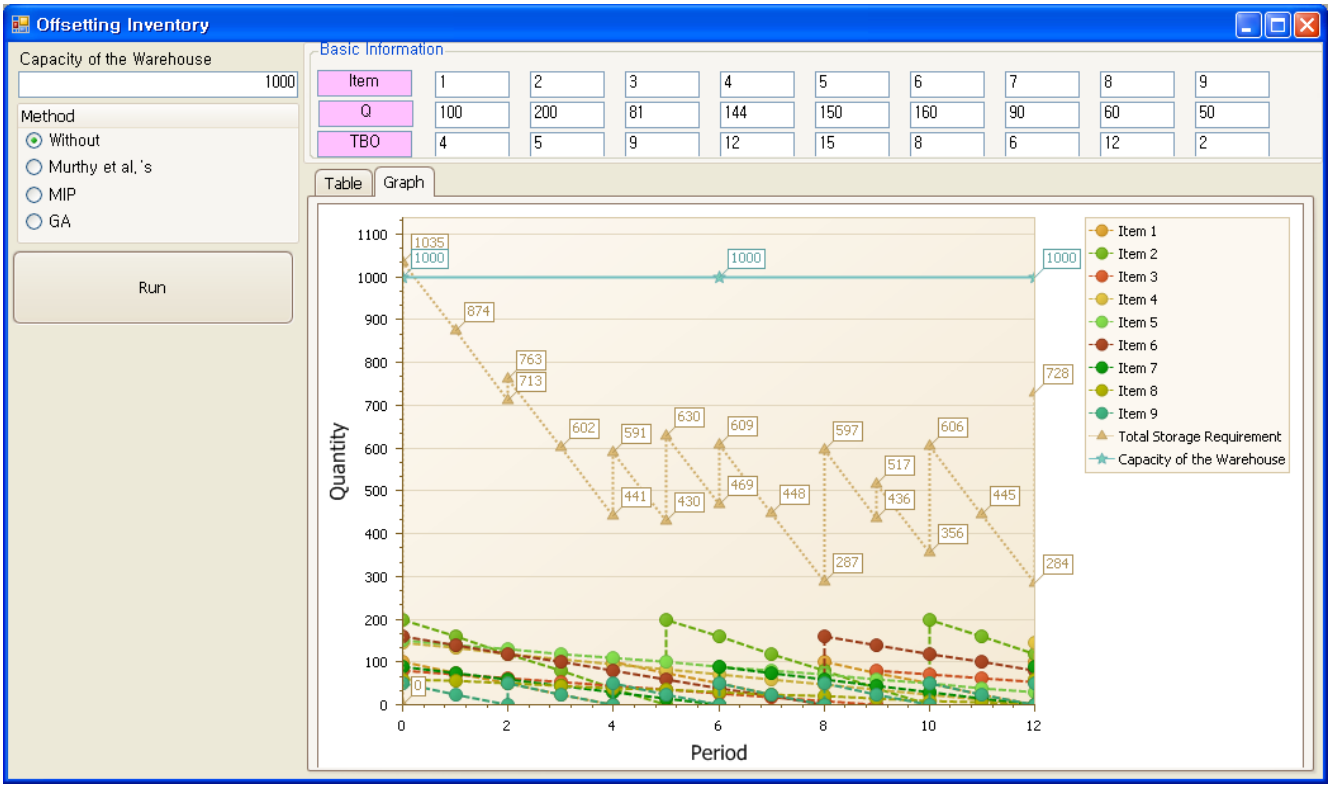

Figure 9: Analysis screen of a warehouse management system

7. CONCLUSIONS

Murthy et al. (2003) dealt with the offsetting that involves minimizing the maximum storage space requirement. This is an important issue with regard to the utilization of the storage space and other resources at warehouses and distribution centers. They formulated the problem and proposed a heuristic algorithm. In this paper, a mixed integer programming model to obtain the optimal solution has been proposed. Further, we develop a genetic algorithm to solve the problem within a rational time interval. Our mixed integer programming model and the genetic algorithm yielded better results than the algorithm of Murthy et al. (2003). We also extended our study to the finite time horizon, which is more general and realistic than the infinite time horizon. The JRP has many potential applications in the supply chain of warehouse, distribution center, and retailer for the multiple items. But if the solution methodology of the general JRP is applied to the real world, the storage space can be inefficiently used. To apply the efficient replenishment policy to the real world, we firstly decide the times between order of each item which minimize the total relevant cost by the solution methodology of the general JRP. Then we have to calculate the first replenishment quantity of each item by the proposed algorithms. We can obtain a good performance of the genetic algorithm for large size problems. Our research may be applied to solve the more general problem of selecting ordering policies to minimize the combined holding, ordering, and storage costs.

8. ACKNOWLEDGEMENTS

The authors are grateful to the constructive review from anonymous referees. This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (The Regional Research Universities Program/Research Center for Logistics Information Technology).

9. REFERENCES

1. Arkin, E., Joneja, D. and Roundy, R. (1989). Computational complexity of uncapacitated multi-echelon production planning problems. Operations Research Letters, 8; pp 61-66.
2. Banerjee, A. and Banerjee S. (1994). A coordinated order-up-to inventory control policy for a single supplier and multiple buyers using electronic data interchange. International Journal of Production Economics, 35; pp 85-91.

3. Cha, B., Moon, I. and Park, J. (2008). The joint replenishment and delivery scheduling problem of the one-warehouse n-retailer system. Transportation Research Part E,44; pp 720-730.
4. Goyal, S. (1975). Analysis of joint replenishment inventory systems with resource restriction. Operations Research Quarterly, 26; pp 197-203.
5. Goyal, S. and Deshmukh, S. (1993). A note on the economic ordering quantity for jointly replenished items. International Journal of Production Research, 31; pp 2959-2961.
6. Hong, S. and Park Y. (2006). A comparison study on retailer-managed and vendor-managed inventory policies in the retail supply chain. Journal of the Korean Institute of Industrial Engineers, 32; pp 382-392.
7. Hoque, M. (2006). An optimal solution technique for the joint replenishment problem with storage and transport capacities and budget constraints. European Journal of Operational Research, 175; pp 1033-1042.
8. Kaspi, M. and Rosenblatt, M. (1983). An improvement of Silver's algorithm for the joint replenishment problem. IIE Transactions, 15; pp 264-269.
9. Kaspi, M. and Rosenblatt, M. (1991). On the economic ordering quantity for jointly replenished items. International Journal of Production Research, 29; pp 107-114.
10. Khouja, M. and Goyal, S. (2008). A review of the joint replenishment problem literature: 1989-2005. European Journal of Operational Research, 186; pp 1-16.
11. Kim, T., Hong, Y. and Chang S. (2006). Joint economic production-shipment policy in a single-manufacturer multiple-item system. International Journal of Industrial Engineering, 13; pp 357-363.
12. Monthatipkul, C. and Yenradee, P. (2007). Positioning safety stocks in a one-warehouse multi-retailer supply chain controlled by optimal inventory/distribution plan. International Journal of Industrial Engineering, 14; pp 169-178.
13. Moon, I. and Cha, B. (2006). The joint replenishment problem with resource restrictions. European Journal of Operational Research, 173; pp 190-198.
14. Moon, I., Cha, B. and Goyal, S. (2008). The joint replenishment problem involving multiple suppliers offering quantity discounts. International Journal of Systems Science, 39; pp 629-637.
15. Murthy, N., Benton, W. and Rubin, P. (2003). Offsetting inventory cycles of items sharing storage. European Journal of Operational Research, 150; pp 304-319.
16. Silver, E.A. (1975). Modifying the economic order quantity (EOQ) to handle coordinated replenishment of two or more items. Production & Inventory Management, 16; pp 26-38.
17. Silver, E.A. (1976). Simple method of determining order quantities in jointly replenishments under deterministic demand. Management Science, 22; pp 1351-1361.
18. Wildeman, R., Frenk, J. and Dekker, R. (1997). An efficient optimal solution method for the joint replenishment problem. European Journal of Operational Research, 99; pp 433-444.

BIOGRAPHICAL SKETCH

Ilkyeong Moon is a Professor of Industrial Engineering at Pusan National University in Korea. He received his B.S. and M.S. in Industrial Engineering from Seoul National University, Korea, and Ph.D. in Operations Research from Columbia University. He currently serves on the editorial boards of several international journals and Editor-in-Chief of *Journal of the Korean Institute of Industrial Engineers.*

Byung-Chul Cha is currently a senior member of Postal Technology Research Center in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. He received the B.S., M.S., and Ph.D. degrees in industrial engineering from Pusan National University, Busan, Korea, in 1995, 1997, and 2005, respectively. He has been a CPIM since 2004. His research interests include SCM and system analysis and design for Korea Post.

Sun-Kwon Kim is currently a research and development engineer of Technical Center in S&T Daewoo Co., Ltd, Busan, Korea. He received his B.S. in Industrial Engineering from Kyungsung University, and M.S. in Industrial Engineering from Pusan National University, Korea. His research interests include SCM and inventory management and design for dampers.