Computers & Industrial Engineering 62 (2012) 29–38

Contents lists available at SciVerse ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A genetic algorithm for minimizing total tardiness/earliness of weighted jobs in a batched delivery system [☆]

Amir Hamidinia, Sahand Khakabimamaghani [*], Mohammad Mahdavi Mazdeh, Mostafa Jafari

Industrial Engineering Dept., Iran University of Science and Technology, Narmak, Tehran, Iran

A R T I C L E   I N F O

Article history:
Received 22 February 2011
Received in revised form 11 August 2011
Accepted 20 August 2011
Available online 27 August 2011

Keywords:
Scheduling
Single-machine
Weighted tardiness/earliness
Batched delivery system
Genetic algorithm

A B S T R A C T

This paper endeavors to solve a novel complex single-machine scheduling problem using two different approaches. One approach exploits mathematical modeling, and the other is based upon genetic algorithms. The problem involves earliness, tardiness, and inventory costs and considers a batched delivery system. The same conditions might apply to some real supply chains, in which delivery of products is conducted in a batched form and with some costs. In such delivery systems, the act of buffering the products can have both positive effects (i.e., decreasing the delivery costs and early jobs) and negative ones (i.e., increasing the number of tardy and holding costs). Accordingly, the proposed solution takes into account both effects and tries to find a trade-off between them to hold the total costs low. The suggestions are compared to existing solutions for older non-batched systems and have illustrated outperformance.

1. Introduction

Nowadays, manufacturers endeavor to schedule their production activities in the best possible way in order to be able to meet the due dates of their customers' orders. This enables them to consider not only delay penalties, but also the penalty of jobs completed before due date in the form of inventory costs. The other issue accounted for in scheduling is the delivery costs of the jobs or batches of jobs. In other words, manufacturers try to minimize the total tardiness/earliness costs besides the delivery costs of the customer orders. Since problems that consider the total tardiness/earliness are closer to real conditions than the problems taking into account the number of tardy/early jobs, here the former type of problems is investigated.

In minimizing the total tardiness/earliness in a single-machine context, it is assumed that in each point of time only one job could be processed. Furthermore, each job has a particular processing time, due date, and weight, which is its earliness/tardiness penalty. A job is called tardy when it is delivered after its due date, and it is called early if its delivery occurs before its due date. This paper aims at solving the problem of minimizing the total tardiness/earliness costs of weighted jobs plus their batched delivery costs when all the jobs are available at time zero and preemption is not allowed. The simple form of this problem is considered in many literatures, but here a batched delivery system, in which the jobs of a customer could be delivered to it in one or many batches, is assumed. In such a delivery system, a job might have different completion and delivery/rendition times (always, rendition time is greater than or equal to the completion time). Each batch might contain at least one or at most all of the jobs of a particular customer, so the number of batches would be equal to at least the number of customers and at most the total number of jobs.

As mentioned above, rendition of a job always occurs after its completion. For investing whether a job is earliness or tardiness, its rendition time should be compared with its due date, and for questing the time needed to hold a job in manufacturer's store, one should take completion time into account with rendition time. Although using the batched delivery system decreases delivery costs considerably, it should be noted that in this method all the jobs assigned to a specific batch should wait until the last job of that batch to be completed before they would be ready for delivery. Naturally, this rule increases the probability of tardiness and so adds to the delay costs.

Accordingly, a batched delivery system decreases the delivery costs while increasing the delayed/early delivery penalties and inventory costs. This paper is an effort to establish a balance between these two factors while holding their sum as small as possible through an appropriate scheduling. For this aim, two different approaches are provided and compared. One is a traditional integer programming solution, and the other is a genetic algorithm solution.

[☆] This manuscript was processed by Area Editor T.C. Edwin Cheng.
[*] Corresponding author. Tel.: +98 912 341 1685.
E-mail addresses: amir.hamidinia.iust@gmail.com (A. Hamidinia), sahand.cs82@gmail.com (S. Khakabimamaghani), mazdeh@iust.ac.ir (M.M. Mazdeh), jafari@iust.ac.ir (M. Jafari).

The remaining of this paper is organized as follows. The next section provides some primary definitions and a review on the literature about the field. The third section is dedicated to the proposed approaches, and the forth one describes the experiments comparing the proposed solutions. The paper concludes in the fifth section.

2. Definitions and literature review

In this section, first, the problem of the paper is described in detail. Second, a review on the current integer programming approaches for solving similar problems is provided. Furthermore, there could be found a brief description of the concept of genetic algorithm. Finally, the state-of-the-art literature on genetic algorithm approaches to the problem under investigation is reviewed.

2.1. The problem

There are N jobs without any ready time and available at the time zero. The jobs belong to F customers each of which has n_j ($1 \leqslant j \leqslant F$) orders (i.e., jobs). Each job has a processing time P_{ij}, a completion time C_{ij}, a delivery deadline or due date d_{ij}, a rendition time R_{ij}, a delay penalty (i.e., cost) α_{ij}, a cost of holding in manufacturer's store h_{ij}, and earliness penalty β_{ij}, in all of which i indicates the job number and j refers to a customer (see Table 1 for more details). A job is called early when its rendition time is before its due date (i.e., $R_{ij} \leqslant d_{ij}$) and on time when it is delivered in its due date (i.e., $R_{ij} = d_i$). Otherwise, it is called a tardy or delayed job. In this problem, no preemption or setup times are considered. Each job can be submitted to the customer as soon as it is processed or its delivery can be postponed to be conducted in a batch of other jobs from the same customer. Deciding between these two options requires simultaneous consideration of amounts of delivery, earliness, tardiness, and holding costs. Moreover, it should be noted that the rendition time of a job is equal to the rendition time of the batch containing that job, because the delivery of all jobs of a batch is postponed to the completion of the last job of that batch. Therefore, the rendition for each job is greater than or equal to its completion time (i.e., $R_{ij} \geqslant C_{ij}$).

With respect to above, there are three types of penalty considered in this paper. These are costs of holding products in manufacturer's stores, earliness costs, and tardiness costs. The first type of penalties is taken into account when the job is completed before it is rendered or $C_{ij} < R_{ij}$. The second type is for when the job is delivered before its due date or $R_{ij} < d_{ij}$. The last type of penalties applies to jobs delivered after their due times (i.e., $d_{ij} < R_{ij}$). Thus, a job may impose one of these types of penalty or two of them simultaneously regarding its completion, due, and rendition times. Please refer to Table 2 to find different possible cases for a job to take penalties.

Table 1
Description of variables of the problem.

Variable	Description	Values taken
F	Total number of customers	
n_j	The number of orders of the jth customer	1, 2, 3,...
N	Total number of all ordered jobs $\left(\sum_{j=1}^{F} n_j = N\right)$	
α_{ij}	Delay cost of the ith job of the jth customer	1, 2, 3,...
β_{ij}	Earliness cost of the ith job of the jth customer	1, 2, 3,...
h_{ij}	Holding (inventory) cost of the ith job of the jth customer	1, 2, 3,...
P_{ij}	Process time of the ith job of the jth customer	1, 2, 3,...
d_{ij}	Delivery time of the ith job of the jth customer	1, 2, 3,...
C_{ij}	Completion time of the ith job of the jth customer	1, 2, 3,...
R_{ij}	Rendition time of the ith job of the jth customer	1, 2, 3,...

Table 2
Various possible cases for a job.

Relation between C, d, and R	Earliness penalty	Tardiness penalty	Holding penalty
$d = C = R$	0	0	0
$d < C = R$	0	$\alpha \times (R - d)$[a]	0
$d < C < R$	0	$\alpha \times (R - d)$	$h \times (R - C)$[b]
$C < d = R$	0	0	$h \times (R - C)$
$C = d < R$	0	$\alpha \times (R - d)$	$h \times (R - C)$
$C < d < R$	0	$\alpha \times (R - d)$	$h \times (R - C)$
$C = R < d$	$\beta \times (d - R)$[c]	0	0
$C < R < d$	$\beta \times (d - R)$	0	$h \times (R - C)$

[a] α indicates the tardiness penalty the manufacturer should pay to the customer for each time unit.
[b] h indicates the cost of holding the job in manufacturer's store for one time unit.
[c] β indicates the earliness penalty the manufacturer should pay to the customer for each time unit.

In addition to these penalties, there is a delivery cost for each batch, which depends on the receiver customer and is unique for each customer. This delivery cost for customer j is identified by variable D_j.

To conclude, this paper aims at minimizing the sum of delivery, delay, earliness, and holding costs. This summation can be shown as $1\|\sum \alpha_{ij}T_{ij} + \beta_{ij}E_{ij} + h_{ij}H_{ij} + D_jY_{jk}$ regarding the common form introduced by Graham, Lawler, Lenstra, and RinnooyKan (1979). In this term, the following new items could be seen:

T_{ij}: $\max\{0, (R_{ij} - d_{ij})\}$.
E_{ij}: $\max\{0, (d_{ij} - R_{ij})\}$.
H_{ij}: $R_{ij} - C_{ij}$.
Y_{ik}: a binary variable indicating whether the kth batch is dedicated to the jth customer or not.

2.2. Integer programming approaches to the problem

In some of the papers in the area of minimizing the total earliness/tardiness, it is assumed that earliness and tardiness should be of degree one and two, respectively, because when a job is early, the customer only misses the inventory costs of the early products, but when it is tardy, not only the production plan of the customer is delayed and interrupted, but also the prestige of the manufacturer is damaged. Gupta and Sen (1983), Sen, Dileepan, and Lind (1995), Su and Chang (1998) and Schaller (2002) are some examples of researchers with this viewpoint to the earliness and tardiness costs, which consider the problem $\sum\left(E_j + T_j^2\right)$, where E_j is the amount of the earliness of the jth job and T_j is the amount of its tardiness.

In other papers, both the tardiness and the earliness are supposed of degree one, and instead, a specific penalty is assigned to each of them. For example, Ying (2008) investigates problem $1|d_j = d|\sum(\alpha_j E_j + \beta_j T_j)$. Furthermore, Garey, Tarjan, and Wilfong (1988), Kim and Yano (1994), Feldmann and Biskup (2003) and Schaller (2007) consider problem $1\|\sum(E_j + T_j)$. Baker and Scudder (1990) provide an excellent survey of the initial work on early/tardy scheduling.

Generally, in the problem surveyed in this paper, two states are assumed: (1) earliness is not acceptable for the customer and is charged by a penalty (i.e., Just-In-Time Logistic) and (2) earliness is fruitful for customer and is promoted by donating a prize regarding the amount of earliness. While the proposed solution is able to cover both the cases (i.e., the second case could be invested easily by assigning minus weights to early jobs), in this paper the second case is not attended.

Another novel aspect of the problem suggested in this paper is the consideration of batched delivery system in it. Potts and Kovalyov (2000) have provided a comprehensive review on the scheduling problems when the jobs are processed or delivered in batches. They specifically have focused on the dynamic programming approaches to this type of problem. Furthermore, Mason and Anderson (1991) have investigated the weighted flow time minimization problems in batch delivery systems and proposed a branch and bound solution for those. Hall and Potts (2003) have provided dynamic programming solutions for a range of scheduling problems with batched delivery systems. Finally, MahdaviMazdeh, Sarhadi, and Hindi (2007) have suggested branch and bound algorithms for weighted sum of flow times in a batched delivery system, when all jobs are available at the time zero and in the presence of ready time (MahdaviMazdeh, Sarhadi, & Hindi, 2008). They have compared their solutions with Hall and Potts's (2003) ones.

The only issue that is not considered in this paper is setup time. Setup time is ???. For examples of works involving this issue, reader is referred to Suriyaarachchi and Wirth (2004) and Supithak, Liman, and Montes (2010).

Since the problem of minimizing the total earliness/tardiness costs is NP-Hard (Hall, Kubiak, & Sethi, 1991), the under investigation problem, which is even more complicated, is NP-Hard too. So, evolutionary optimization solutions would be appropriate for approaching such problems. The next sections give insight into the genetic algorithms provided in the literature for resolving these NP-Hard scheduling problems.

2.3. Genetic algorithm: a brief definition

Genetic algorithm is an evolutionary algorithm, which is based upon Darwin's theory about evolution. It evolves an initial set of solutions in order to reach an optimized solution. "Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by the expectation, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are chosen according to their fitness – the more suitable they are, the more chance they have to involve in the reproduction. This is repeated until some condition (e.g., reaching the maximum number of populations or no improvement in the fitness of the best solution) is satisfied" (Obitko, 1998). Accordingly, a genetic algorithm can be overviewed as a four-step process: initialization, selection, reproduction, and termination. Each of these steps and other significant concepts of a genetic algorithm (i.e., representation scheme and fitness function) are elaborated in Section 3.

2.4. Genetic algorithm approaches to the problem

Scheduling problems belong to the class of strongly NP-Hard problems (Pinedo, 1995). According to this fact and because genetic algorithms (GA) are proven appropriate for diverse optimization problems (Davis, 1991; Goldberg, 1989), some researchers have exploited these algorithms to solve different types of scheduling problems including earliness/tardiness problems.

For example, in (Valente & Gonçalves, 2009), this type of scheduling problem is attended and the tardiness cost is considered in power of two. A genetic algorithm based upon a random key alphabet U(0,1) (by Bean (1994)) is proposed for this problem in that paper. Furthermore, several genetic algorithms based on this approach are presented, which differ on the generation of the initial population, as well as on the use of local search. The proposed procedures are compared with existing heuristics and with optimal solutions for the smaller instance sizes.

Moreover, Koksalan and BurakKeha (2003) used a GA to solve two bi-criteria scheduling problems: minimizing flow time and maximum earliness, and minimizing flow time and number of tardy jobs. Furthermore, Hallah (2007) applied hybrid of SA and GA to minimize total earliness and tardiness on single machine, and he tested the capability of the proposed algorithm. Jolai et al. (2007), also, proposed a genetic algorithm for bi-criteria single machine scheduling problem of minimizing maximum earliness and number of tardy jobs. For this problem, they developed a genetic algorithm by exploiting its general structure that further improves the initial population, utilizing a heuristic algorithm on the initial population.

Besides single machines, GA is also applied by Cheng, Gen, and Tozawa (1995) for scheduling in identical parallel machine systems. They are aimed at minimizing the maximum weighted absolute lateness.

In most of these researches, it is assumed that the products are delivered to customers instantly after they are processed or they are buffered in batches with pre-specified sizes, which are delivered to customers after being completely filled. In this paper, some new and different assumptions are considered. Here, it is supposed that each job might be delivered to the corresponding customer instantly after completion of its process, or it might be buffered in batches with flexible sizes, each of which belongs to a particular customer. Therefore, the number of batches could vary between the number of customers and the total number of jobs. In other words, this paper aims at holding a trade-off between delivery, tardiness, earliness, and holding costs and finally minimizing the sum of them.

3. Proposed solutions

This section contains two parts, each of which presents one different solution to the discussed problem. The first part is dedicated to the suggested integer programming approach, and the second one elaborates on the proposed genetic algorithm.

3.1. Proposed integer programming solution

Before proposing the mathematical model, the chief idea of the solution is described generally. Initially, N (i.e., the total number of jobs) empty batches are supposed, each of which has a number showing their position in the sequence of batches. In other words, the jobs of batch number $k-1$ are processed and delivered earlier than the kth batch's jobs.

Since each batch should only contain the jobs of a particular customer and a customer might have all of its orders inside a single batch, the number of jobs at each batch could be at most equal to the maximum number of jobs belonging to a customer, which we name m here. Accordingly, each batch is divided into m parts. For example, if there are two customers with 3 and 5 orders, then 3 + 5 = 8 batches each with 5 = max{3,5} parts should be supposed initially (see Fig. 1).

In addition to the batches, each part inside a batch has a priority number, which indicates the order of processing the job inside it. In other words, the order of processing the jobs inside a particular batch is determined by the order of insider parts. For example, in Fig. 1, the second order of customer 1 is processed after its first order, because it is assigned to part 4 of the first batch, while the first order is assigned to the first part of that batch. This order of processing is important because the difference between job's completion and rendition times determines its holding (i.e., inventory) costs.

After the number of batches and parts inside them is decided, the jobs are assigned to the parts randomly but controlled by some constraints. These constraints prevent jobs of different customers to be assigned to the same batch, which is one of the rules of job allocation. Naturally, at last some of the initial hypothetical empty

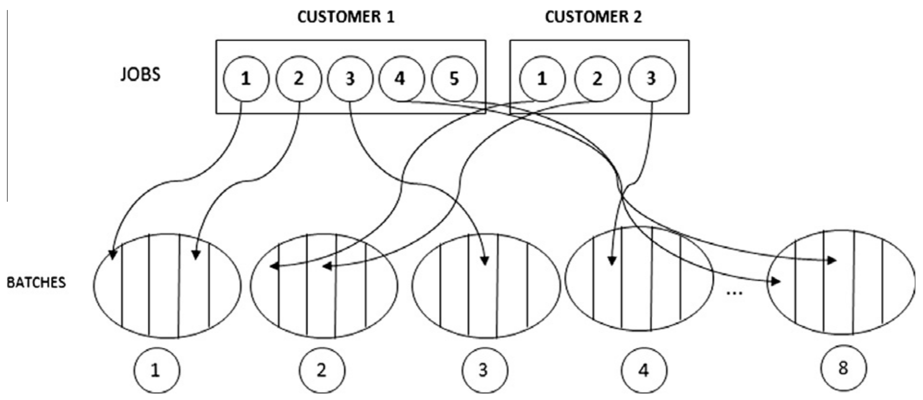


Fig. 1. A sample job allocation.

batches would remain empty with no jobs assigned to them (e.g., batches 5–7 in Fig. 1). These would be eliminated from the list of batches. Fig. 1 illustrates the overall process of a sample job allocation for the mentioned example.

After the assignment step, regarding the batch and part numbers, that reveals the order of processing and rendering the jobs, the completion and rendition times of the jobs and consequently whether each job is tardy, early or on-time would be identified. As addressed earlier, the rendition time of each job equals the completion time of the batch in which that job is placed, because each job should wait for the last job belonging to the same batch to be completed. After identifying the tardy and early jobs, the total costs, including the delivery, delay, earliness, and holding costs, can be calculated.

With respect to above description, the following integer programming model is proposed:

$$Min\sum_i\sum_j \alpha_{ij}\cdot\max\{0,R_{ij}-d_{ij}\}+\sum_i\sum_j h_{ij}\cdot(R_{ij}-C_{ij}) + \sum_i\sum_j \beta_{ij}\cdot\max\{0,d_{ij}-R_{ij}\}+\sum_j\sum_k D_j\cdot y_{jk}+\sum_k M\cdot Q_k \tag{1}$$

ST:

$$\sum_k\sum_{k'} x_{ijkk'}=1 \quad \forall i\in(1,2,\ldots,n_j),\ \forall j\in(1,2,\ldots,m) \tag{2}$$

$$\begin{cases} -(\sum_i\sum_j x_{ijkk'}-\sum_i x_{ijkk'})+M\cdot Q_k\geqslant 0\\ (\sum_i\sum_j x_{ijkk'}-\sum_i x_{ijkk'})+M\cdot(1-Q_k)>0 \quad \forall j\in(1,2,\ldots,m),\\ \quad \forall k\in(1,2,\ldots,n),\ \forall k'\in(1,\ldots,l)\end{cases} \tag{3}$$

$$\begin{cases} -\sum_i x_{ijkk'}+M\cdot y_{jk}\geqslant 0\\ \sum_i x_{ijkk'}+M\cdot(1-y_{jk})>0 \quad \forall j\in(1,2,\ldots,m),\\ \quad \forall k\in(1,2,\ldots,n),\ \forall k'\in(1,\ldots,l)\end{cases} \tag{4}$$

$$\left\{\begin{array}{l} C_k=\sum_{i=1}^{n_j}\sum_{j=1}^{m}\sum_{k'=1}^{l} x_{ijkk'}\cdot P_{ij}+C_{k-1} \quad \forall(k>1)\in(1,2,\ldots,n)\\ C_1=\sum_i\sum_j\sum_{k'} x_{ij1k'}\cdot P_{ij}\end{array}\right\} \tag{5}$$

$$R_{ij}=\sum_{k=1}^{m} x_{ijkk'}\cdot C_k \quad \forall i\in(1,2,\ldots,n_j),\ \forall j\in(1,2,\ldots,m),\ \forall k'\in(1,\ldots,l) \tag{6}$$

$$\begin{cases}\begin{cases}\text{if } x_{ijkb'}=1\\ \text{then } C_{ij}=C_{k-1}+\sum_i\sum_j\sum_{k'=1}^{b'} x_{ijkk'}\cdot p_{ij} \quad \forall i\in(1,2,\ldots,n_j),\\ \quad \forall j\in(1,2,\ldots,m),\ \forall k\in(1,2,\ldots,n),\ \forall b'\in(1,2,\ldots,l)\end{cases}\\ C_0=0\end{cases} \tag{7}$$

$$x_{ijk},y_{jk},Q_k:\mathit{Binary} \quad \forall i\in(1,2,\ldots,n_j),\ \forall j\in(1,2,\ldots,m),\ \forall k\in(1,2,\ldots,n) \tag{8}$$

Table 3 provides a reference for the variables found in the above model.

As addressed earlier, there should be some constraints preventing jobs of different customers to be assigned to the same batch. This constraint is exerted by adding relation number (3) to the model. On the other hand, the batches that remain empty after the allocation process are discarded regarding relation number (4). More precisely, the proposed model consists of the following equalities/inequalities (please refer to the numbers in the model to see the corresponding constraint):

(1) This equation is the objective function of the problem. The first term sums the delay costs, the second one is the holding costs of the completed jobs in the manufacturer's stores, the third term indicates the sum of earliness penalties, and the fourth one shows the total delivery costs. The last part of the objective function adds a penalty to this function and is described in paragraph number (3) below.

(2) This equation ensures that each job is assigned only to one part of one particular batch. For example, if the 2nd job of the 1st customer is placed in the 1st part of the 3rd batch or $x_{2131}=1$, consequently all $x_{21kk'}$ when $k\neq 3$ or $k'\neq 1$ is equal to 0.

(3) If a job belonging to a particular customer is assigned to a specific batch, that batch cannot include any jobs belonging to other customers. In other words, each non-empty batch should be dedicated to only one customer. This constraint is applied using a big penalty, such that if the kth batch is related to more than one customer, then the variable Q_k equals 1 and the mentioned penalty (i.e., big number M) is considered in the objective function. Naturally, according to greatness of this penalty, the solutions including it would be discarded during the optimization process.

(4) Regarding that the initial number of batches equals the total number of jobs (i.e., N), there might remain some empty batches after the job assignment step. Thus, in this constraint, the binary variable y_{jk} is defined such that its value

Table 3
Description of variables of the proposed mathematical model.

Variable	Description	Values taken
n_j	The number of orders of the jth customer	1, 2, 3,...
l	The greatest n_j ($l = \max\{n_1, n_2, \ldots, n_F\}$)	
N	Total number of all ordered jobs $\left(\sum_{j=1}^{F} n_j = N\right)$	
α_{ij}	Delay cost of the *i*th job of the *j*th customer	1, 2, 3,...
β_{ij}	Earliness cost of the *i*th job of the *j*th customer	1, 2, 3,...
h_{ij}	Holding (inventory) cost of the *i*th job of the *j*th customer	1, 2, 3,...
D_j	Delivery cost of batches belonging to the *j*th customer	1, 2, 3,...
$x_{ijkk'}$	Equals 1 if the *i*th job of the *j*th customer is in *k*'th part of the *k*th batch and equals 0 otherwise	0, 1
y_{ik}	Equals 1 if there is a job belonging to the *k*th batch which relates to the *j*th customer and is equal to 0 otherwise	0, 1
Q_k	Equals 1 if some jobs from different customers are assigned to the *k*th batch and is equal to 0 otherwise	0, 1
P_{ij}	Process time of the *i*th job of the *j*th customer	1, 2, 3,...
d_{ij}	Delivery time of the *i*th job of the *j*th customer	1, 2, 3,...
C_k	Completion time of the *k*th batch	1, 2, 3,...
C_{ij}	Completion time of the *i*th job of the *j*th customer	1, 2, 3,...
R_{ij}	Rendition time of the *i*th job of the *j*th customer	1, 2, 3,...
M	A big number	

is 1 if there is any job belonging to *j*th customer in the *k*th batch and, otherwise, it equals 0. This variable helps putting empty batches away when calculating costs.

(5) This equation computes the completion time of each batch, which is equal to the sum of the processing times of the jobs assigned to that batch plus the completion time of the previous batch. The order of batches is indicated by their numbers.

(6) Regarding that each job assigned to a specific batch should wait for all of the jobs belonging to that batch to be completed before it could be delivered, the rendition time of each job is set equal to the completion time of the corresponding batch.

(7) This relation computes the completion time of each job, which is the completion time of the previous batch plus the processing time of this job and every jobs in the same batch which are assigned to it before the current job.

(8) This constraint introduces the variables $x_{ijkk'}$, y_{ik}, and Q_k as binary.

3.2. Proposed genetic algorithm

In designing every genetic algorithm, several concepts should be declared carefully. These include chromosome structure, fitness function, selection method, and genetic operators (i.e., mutation and crossover functions). The following sections introduce the methods we have exploited in our research regarding each issue. Finally, the steps of the proposed genetic algorithm are described.

3.2.1. Chromosome structure

The representation scheme, which is the most important aspect of a GA, should be able to capture every aspects of a possible solution for the corresponding problem. For the problem of this research, a chromosome should determine the products included in each batch and the order of completing them, which would reveal the order and time of delivering the batches. Also, the chromosome representation should consider the limitation that each batch only must contain products belonging to one customer.

With respect to above, we propose the structure illustrated in Fig. 2. As it could be seen, each chromosome consists of a general batch permutation string besides an index vector and an order priority vector for each customer.

In the batch permutation string, each gene points to a particular batch by its number. The repeated batch numbers are not allowed in this string as the name, 'permutation', indicates. Furthermore, the string is constituted by several substrings, each of which is dedicated to a specific customer. The length of each substring equals the number of orders of the corresponding customer. For

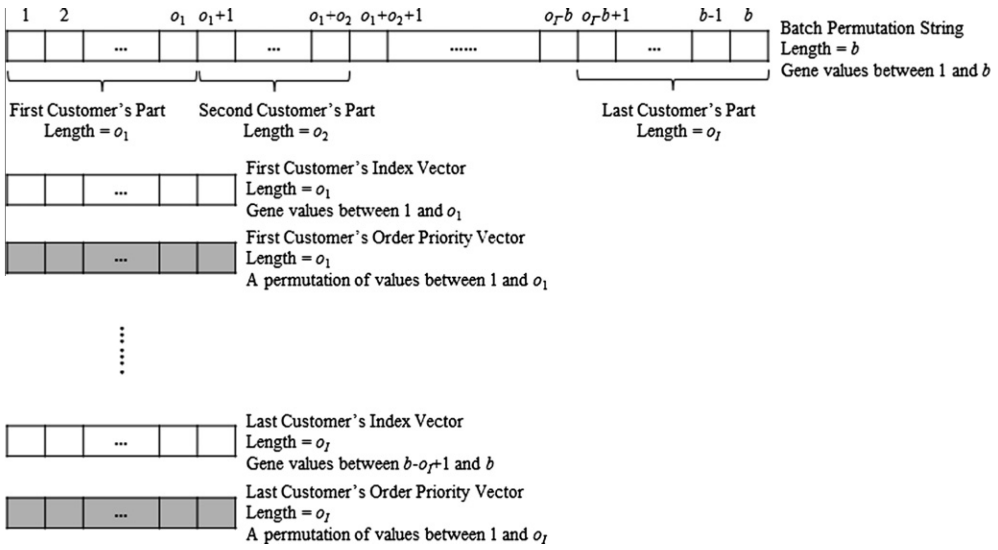

1
2
o1
o1+1
o1+o2
o1+o2+1
oI-b
oI-b+1
b-1
b
Batch Permutation String
Length = b
Gene values between 1 and b
First Customer's Part
Length = o1
Second Customer's Part
Length = o2
Last Customer's Part
Length = oI
First Customer's Index Vector
Length = o1
Gene values between 1 and o1
First Customer's Order Priority Vector
Length = o1
A permutation of values between 1 and o1
Last Customer's Index Vector
Length = oI
Gene values between b-oI+1 and b
Last Customer's Order Priority Vector
Length = oI
A permutation of values between 1 and oI

Fig. 2. Proposed chromosome structure: In this figure, it is assumed that the number of customers is I and the number of *i*th customer's orders is o_i. Furthermore, it is supposed that the summation of numbers of all of the orders equals b, which is the initial number of batches.

example, if the first customer has o_1 orders, then the first o_1 cells of the permutation string belong to it and the batches pointed to by these cells could be used for packing this customer's orders. Then comes the substring of the second customer and so on.

In addition, there is an index vector for each customer. Like the substrings, the length of each index vector equals the number of the corresponding customer's orders. Gene number one is dedicated to order number one, gene number two is dedicated to order number two, and so on. Moreover, each gene in the vector addresses a cell in the customer's substring in the batch permutation string. The batch pointed to by that permutation string cell determines the batch in which that particular order (to which the index vector belongs) should be delivered. The index vector could also contain repeated indexes, which mean that two different orders are places in the same batch.

The third component of the proposed structure is called order priority vector. There is an order priority vector per each index vector with the same length as the corresponding index vector. Each order priority vector is a permutation of numbers between 1 and the length of that vector. Each number belongs to one order of the customer to whom the order priority vector is related and indicates that orders priority in processing. The first cell indicates the priority of the first order, and the second cell indicates the priority of the second order. When two orders of a particular customer are assigned to two different batches, they are processed regarding the priority of their batches. But, when two orders fall in the same batch, it is needed to refer to the order priority vector of the corresponding customer to find the order of processing them. The smaller the amount of the order priority vector cell, the higher the priority of the corresponding job/order.

The next section describes how to interpret a chromosome and translate it to a solution.

3.2.2. Fitness function

Before calculating the fitness of a chromosome, first it should be interpreted and the solution that it suggests should be derived. As stated in the previous section, there is an index vector for each customer in each chromosome with the length of that customer's orders. For finding each order's corresponding batch regarding the values stated in the chromosome, first we should refer to the gene of that order in the index vector of the customer who has made it. The value of that gene leads us to a cell in the batch permutation string. Finally, the number of batch assigned to the order could be read from that cell in the batch permutation string.

When there are more than one job assigned to a particular batch, all of which should belong to one specific customer, all one has to do to find the sequence of processing them, is to refer to order priority vector of that customer in the chromosome. Regarding the values of cells in order priority vector related to those jobs (i.e., orders), the processing priority of each job could be found as stated earlier in Section 3.2.1 (smaller cell value means higher priority). If the processing of a job in a batch is completed earlier than that batch's completion time, which means that it possesses higher priority than at least one job in that batch, it adds some holding (inventory) costs to the total costs (refer again to Fig. 2).

Now that the jobs and customers of each batch are determined, it is possible to construct a solution. For doing this, first empty batches are excluded from the batch sets. Then, the occupied batches are sorted regarding their numbers, and the jobs inside each batch are ordered with respect to their priority. The resulted sequence indicates the order of completing the jobs and delivering the batches. Consequently, the delivery and delay costs can be calculated as described in Section 2, and the total cost is the fitness of the current individual. Fig. 3 provides an example of a chromosome and its fitness, which represents a solution for a 3-customer 6-job problem.

Now that the way of interpreting a chromosome is described we return back to the requirements and restrictions stated in Section 2 to see whether the designed structure satisfies them or not. First, in this structure, some of the batches might be filled while the others are empty, so the occupied batches are declared and we ignore the empty batches in our calculations. Second, it is exactly determined that which batch is dedicated to each job/order as described above. Third, since the order of executing the batches is as their numbers indicate (e.g., batch number 10 would be completed after batches number 1–9), we know the order of completing and/or delivering each batch at the end. Finally, since we partition the batch permutation string and dedicate each substring to a particular customer, no customer can use the batches of others and, consequently, no batch will contain jobs from more than one customer. So the proposed structure satisfies all the conditions and constraints besides providing the possibility to cover the whole search space.

3.2.3. Selection method

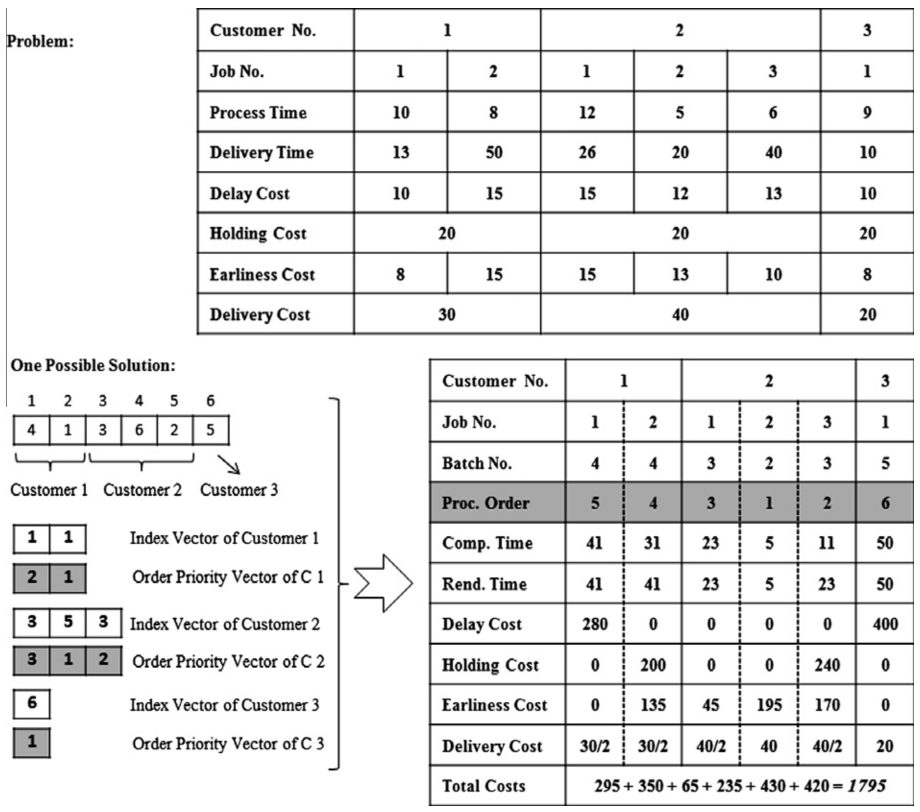
Selection methods are generally grouped into four categories: Roulette Wheel, Rank, Steady-State, and Elitism (Obitko, 1998). Among these, our method is more close to Steady-State method. In this method, the big part of individuals survives to the next generation. Then, a few of the best (i.e., fittest) chromosomes are selected for generating new offspring through genetic operators described in the next section. Then, the weaker individuals (i.e., with lowest fitness) are replaced with the better newly generated individuals.

In our proposition, after sorting the chromosomes regarding their fitness in ascending order, top $\lfloor \sqrt{n} \rfloor$ individuals are selected for crossover. By selecting this number of the best chromosomes, it is possible to generate about n new offsprings, which increases the exploration of the algorithm by providing more inputs to the mutation operator, besides augmenting the size of generated population. Then, every new chromosome will be mutated with chance of 50%. Consequently, the size of current population will increase to about $2.5n$ (i.e., n initial + n offsprings of crossover $+n/2$ offsprings of mutation). Among these individuals, n fittest ones are transferred to the next iteration. Please refer to Section 3.2.5 to view the overall description of the proposed GA and the selection method.

3.2.4. Genetic operators

There exist two kinds of genetic operators for reproduction. One of them is crossover function, which combines two selected chromosomes in order to generate two new offspring containing some of the features of their parents. There have been many types of crossover functions proposed in the literature (e.g., one point, two point, uniform, etc.). The detailed description of these types is out of the scope of this paper. Here, the simplest type of them, the one-point crossover, is chosen for combining index vectors. In this function, first, both of the parent chromosomes' index vectors are divided into two equal parts. Then, the first parts from the first parent and the first parts from the second parent are copied into the first parts of the first and second offsprings, respectively. Then, the second parts from the first parent and the second parts from the second parent are copied to the second parts from the second and first offsprings, respectively. Fig. 4 illustrates this type of crossover function.

Regarding that the batch permutation string and order priority vectors are permutation and should not contain repeated numbers, a crossover operator designed for permutation-based representations should be exploited for them. One of the well-known methods for this aim is *position-based crossover,* which is proposed by Syswerda (1989). A slightly modified version of his method is used in the proposed genetic algorithm. This crossover operator consists of three main steps:



Problem:

Customer No.	1		2			3
Job No.	1	2	1	2	3	1
Process Time	10	8	12	5	6	9
Delivery Time	13	50	26	20	40	10
Delay Cost	10	15	15	12	13	10
Holding Cost	20		20			20
Earliness Cost	8	15	15	13	10	8
Delivery Cost	30		40			20

One Possible Solution:

1 2 3 4 5 6
4 1 3 6 2 5
Customer 1
Customer 2
Customer 3
1 1 Index Vector of Customer 1
2 1 Order Priority Vector of C 1
3 5 3 Index Vector of Customer 2
3 1 2 Order Priority Vector of C 2
6 Index Vector of Customer 3
1 Order Priority Vector of C 3

Customer No.	1		2			3
Job No.	1	2	1	2	3	1
Batch No.	4	4	3	2	3	5
Proc. Order	5	4	3	1	2	6
Comp. Time	41	31	23	5	11	50
Rend. Time	41	41	23	5	23	50
Delay Cost	280	0	0	0	0	400
Holding Cost	0	200	0	0	240	0
Earliness Cost	0	135	45	195	170	0
Delivery Cost	30/2	30/2	40/2	40	40/2	20
Total Costs	295 + 350 + 65 + 235 + 430 + 420 = *1795*					

Fig. 3. An example of a problem and one of its possible solutions encoded in a chromosome.

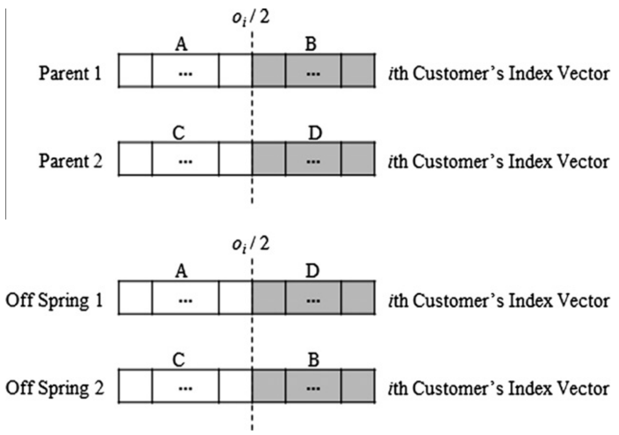

$o_i/2$
A
B
Parent 1
ith Customer's Index Vector
C
D
Parent 2
ith Customer's Index Vector
$o_i/2$
A
D
Off Spring 1
ith Customer's Index Vector
C
B
Off Spring 2
ith Customer's Index Vector

Fig. 4. Crossover operator for index vectors.

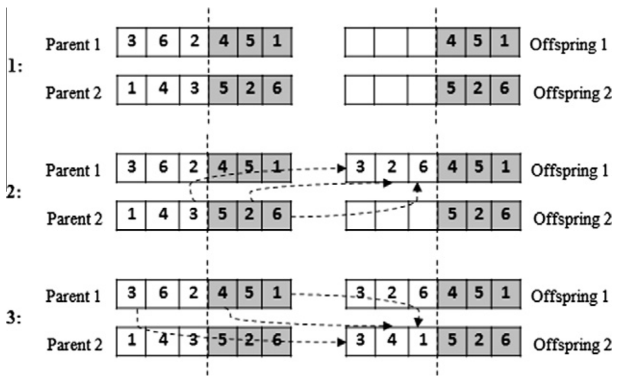


Fig. 5. Crossover operator for permutation string and order priority vectors.

1. Divide the parent strings into two halves and copy the second halves of them to the second halves of the offsprings.
2. For filling in the first half of the first offspring, refer to the second parent and find the values not included in the first offspring's second half currently and copy them into the empty genes in their original order in the second parent.
3. Execute step 2 for the second offspring too, but, this time, refer to the first parent instead of the second one.

Going through these three steps guarantees that there is no duplicate values in the offspring batch permutation strings. Fig. 5 depicts this operator with a simple example. The only difference between the proposed method and Syswerda's one is that he utilize uniform crossover while we use a one-point crossover function. This does not damage the rationale, because in fact one-point crossover is an especial form of uniform crossover.

The second kind of genetic operators is mutation. Mutation increases the algorithm exploration by adding randomly changed individuals to the current population. In the genetic algorithm, when the fitness of the best individual remains the same for several generations, it means being trapped in a local optimum. In such situations, increasing the exploration of algorithm, which depends on the mutation function, might be useful for approaching the globally optimum solution. There has been several mutation functions proposed in the literature, including single-point, double-point, multi-point, inversion, swap, etc. Innovatively, here, we utilize a dynamic mutation for all batch permutation string, index vectors, and order priority vectors. Our experiments show a better performance for this mutation compared with static one-, two- or multi-point operators.

A multi-point mutation mechanism in which the number of mutation points (i.e., p) depends on the number of consecutive generations for which the fitness of the best solution found so far remains the same is used. The amount of p for batch permutation string is calculated through Eq. (9) in which g shows the number of consecutive generations for which the algorithm solution has not improved and b is the number of batches (the length of permutation string):

$$p = \begin{cases} \sqrt{b}, & \left(\frac{g}{50}\right) > \sqrt{b} \\ \frac{g}{50}, & \text{otherwise} \end{cases} \tag{9}$$

For order priority vectors and index vectors, the same strategy is used, but here p is limited to $l/2$ instead of $\sqrt{b}$, where l is the length of corresponding vector. For index vectors, the p mutation points are selected randomly and the value of the chosen genes is replaced with random values in the valid range. For the batch permutation string and order priority vectors, a swap mutation function is utilized, in which two random distinct genes are selected and their values are swapped. This swapping process, which ensures non-repeating values in the arrays, is repeated p times for each array.

3.2.5. Overview of the genetic algorithm

Every genetic algorithm includes four chief steps, namely initialization, selection, reproduction, and termination. The following pseudo-code depicts these main steps.

```
Inputs: the orders of customers, $n$ (the number of initial
  individuals), $I$ (the number of iterations), $p$ (mutation
  probability)
Outputs: the best solution
//Initialization
create $n$ chromosomes
for each chromosome
set values of ‘Batch Permutation Strings’ of the chromosomes
  to random permutations
set values of ‘Index Vector’ cells of the chromosomes to
  random amounts between 1 and $o_i$
set values of ‘Order Priority Vectors’ of the chromosomes to
  random permutations
construct a solution for each chromosome based on its values
  using train set and measure itsfitness
//Iterations
for $i$ from 1 to $I$ do
  //Selection
  sort the individuals based on their fitness
  select top $\lfloor\sqrt{n}\rfloor$ chromosomes for crossover
  //Reproduction
  for each pair in $\lfloor\sqrt{n}\rfloor$ selected individuals
conduct the crossover operator to generate two offsprings
  select $p$ percent of new chromosomes randomly for
  mutation
mutate each individual selected for mutation
  for each about $(1 + p) \times n$ new individuals
    construct the solution
    calculate the fitness of the solution
  sort all the $(2 + p) \times n$ chromosomes regarding their fitness
  and select the top $n$ of them for the next generation
//End of iteration
select the fittest individual as the output of algorithm
//End of Genetic Algorithm
```

The output of this genetic algorithm would be a chromosome with a low total cost, which represents a solution for the scheduling problem described in Section 2.1.

4. Experiments

Initially, the performance of the proposed genetic algorithm was to be compared with that of integer programming. The proposed algorithm is implemented with C#, and the integer programming is executed using Lingo software application. Although simple examples were used to evaluate algorithms, not oddly, the Lingo was unable to provide an answer to these examples, even when it was ran for a local optimum, because regarding the number of constraints the complexity of the integer programming model was too high. The Lingo took more than 10 h to give outcome and crashed with an error message at last, even for simple example problems. Consequently, it was decided to evaluate the proposed method toward other genetic algorithm proposed for scheduling non-batched delivery system in order to both evaluate the benefits of batched delivery system and the performance of the proposed algorithm. It goes without saying that in this type of problems (i.e., non-batched delivery) there would be no consideration of holding costs, while earliness, tardiness, and delivery costs are applied yet.

In the experiments, two comparison criteria are considered: run time and outcome (i.e., total cost of the final solution). To solidify the experimental results, the approaches are examined in different situations, namely simple and complicated cases. For each case, the genetic algorithm was executed 20 times with different random initializations and the average performance is recorded. The error of genetic algorithm approaches is computed through Eq. (10). In this equation, sol_i is the solution of genetic algorithm in its ith execution. Furthermore, *optimum* refers to the best solution of the genetic algorithm among 20 different outcomes.

$$error = 1/20 \sum_{i=1}^{20} \frac{sol_i - optimum}{optimum} \tag{10}$$

While it is not guaranteed that the genetic algorithm results in the global optimum, but according to the minuscule amount of errors in outcomes and enough time let to the algorithms to run, it could be concluded that the resulted solutions are close to the global optimum. Next sections describe the cases and their results.

4.1. Simple cases

For the simple experiment, two 2-customer 8-job problem was designed. The delivery cost, process time, due date, delivery cost, delay cost, earliness cost, and holding cost of each customer’s orders could be seen in Tables 4 and 5. Tables 6 and 7 illustrate the results of running both the proposed and traditional genetic algorithms for these problems. As seen in these tables, the novel genetic algorithm has surpassed the traditional method very significantly. While the time it takes for the proposed algorithm to reach the answer is several times it takes for non-batched delivery genetic algorithm, which is natural regarding the high number of constraints in the batched delivery scheduling problem, it results in less total costs than the non-batched delivery scheduling. Furthermore, the errors of the both algorithms are zero, which indicates that the algorithms would output the best solutions presented in the table each time they are ran. This error is reached by customizing genetic population size and iteration numbers to the cases.

From the results, it is recognized that batched delivery system causes considerable amounts of saving. Moreover, the proposed genetic algorithm reaches the solution in a reasonable time.

Table 4
Specifications of the simple case a.

Customer No.	1				2			
Job No.	1	2	3	4	1	2	3	4
Process time	10	12	8	10	5	18	13	9
Due date	40	28	80	10	25	32	85	110
Delivery cost (D)	70				50			
Delay cost (α)	10	14	15	9	8	12	10	9
Earliness cost (β)	4	3	5	5	4	4	6	5
Holding cost (h)	5							

Table 5
Specifications of the simple case b.

Customer No.	1				2			
Job No.	1	2	3	4	1	2	3	4
Process time	20	15	18	17	15	18	24	20
Due date	40	68	140	102	25	45	85	125
Delivery cost (D)	140				170			
Delay cost (α)	6	8	5	6	4	6	7	5
Earliness cost (β)	8	6	5	9	4	6	10	7
Holding cost (h)	1							

Table 6
Results of experiments on the simple case a.

Algorithm	Run time (s)	Best solution's cost	Error
Non-batched delivery	0.05	1049	0
Batched delivery	1.05	1007	0

Table 7
Results of experiments on the simple case b.

Algorithm	Run time (s)	Best solution's cost	Error
Non-batched delivery	0.03	1514	0
Batched delivery	1.14	1378	0

4.2. Complicated cases

As a complicated example, a problem with three customers and 20 orders was used. Tables 8 and 9 exhibit the results for these cases. Again in these cases, the proposed genetic algorithm produces a better solution compared with traditional algorithm, which indicates the profits of batched delivery system. But, especially in complicated case b, the difference is not remarkable, because the delivery costs are assumed to be low in these cases. Furthermore, the holding costs are not as small as the previous cases here. Another point is the ignorable amount of error of the proposed algorithm, which indicates its reliability. Looking at the run times, it is understood that the proposed algorithm takes reasonable time to finish considering the traditional algorithm and regarding the complexity of batched delivery system.

Table 8
Results of experiments on the complicated case a.

Algorithm	Run time (s)	Best solution's cost	Error
Non-batched delivery	1.99	7599	0
Batched delivery	331.6	7528	0.012

Table 9
Results of experiments on the complicated case b.

Algorithm	Run time (s)	Best solution's cost	Error
Non-batched delivery	4.24	6293	≈ 0
Batched delivery	55.31	6277	0.032

5. Conclusions

After introducing a new type of scheduling problem with batched delivery system, which is an NP-hard problem, two different solutions are proposed for it. The first is an integer programming approach, and the second is a genetic algorithm. The results of experiments indicate that the current applications, such as Lingo, are unable to implement the proposed integer programming model due to the high number of constraints. Accordingly, it is impossible to compare the proposed genetic algorithm with global optimum or even local optimum resulted from that model. Thus, in order to exhibit the advantages of batched delivery system besides evaluating the performance of proposed genetic algorithm, a traditional genetic algorithm approach to simple (non-batched) delivery scheduling problems is used as the benchmark. Experiment results lead to some conclusions.

First, although in all cases the novel genetic algorithm would outperform (or at least be the same as) the traditional one regarding the total costs, it could be understood that the amount of this outperformance completely depends on the specifications of the problem. That is the higher the delivery costs compared with holding costs, the better the batched deliver scheduling. In other words, batched delivery system would yield more economic savings in circumstances with higher delivery costs and lower holding costs.

Second, it can be seen that the proposed genetic algorithm provides an appropriate tool for scheduling production lines with batch delivery systems, since it reaches reasonable answers in acceptable amounts of time. Furthermore, regarding the low computation time required by the proposed genetic algorithm, it seems suitable for simple real-time applications with similar properties.

As a topic for further research, interested researchers can examine the efficiency of the proposed method in real-time applications such as rocket navigation and CPU scheduling. Moreover, other restricting conditions, such as batch-specific delivery deadlines and limited batch capacity, could be challenging problems that real situations encounter with. So, they are suitable for pondering and examining in future researches too. Another point is the efficiency of other types of selection, cross-over, and mutation operators or other parameter settings such as mutation probability and selection rate compared with the ones chosen in the current research. Finally, one might be interested in providing a more efficient chromosome structure for solving this optimization problem.

References

Baker, K. R., & Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: A review. *Operations Research, 38*, 22–36.
Bean, J. C. (1994). Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing, 6*, 154–160.
Cheng, R., Gen, M., & Tozawa, T. (1995). Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. *Computers & Industrial Engineering, 29*, 513–517.
Davis, L. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.
Feldmann, M., & Biskup, D. (2003). Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering, 44*, 307–323.
Garey, M. R., Tarjan, R. E., & Wilfong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research, 13*, 330–348.
Goldberg, D. E. (1989). *Genetic algorithms in search. Optimization and machine learning*. New York: Addison-Wesley.
Graham, R. L., Lawler, E. L., Lenstra, J. K., & RinnooyKan, A. H. G. (1979). Optimization and approximation in deterministic machine scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Gupta, S. K., & Sen, T. (1983). Minimizing a quadratic function of job lateness on a single machine. *Engineering Costs and Production Economics, 7*, 187–194.

Hall, N. G., Kubiak, G. W., & Sethi, S. P. (1991). Earliness–tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Operations Research, 39*, 847–856.

Hall, N. G., & Potts, C. N. (2003). Supply chain scheduling: Batching and delivery. *Operations Research, 51*, 566–584.

Hallah, R. M. (2007). Minimizing total earliness and tardiness on a single machine using hybrid heuristic. *Computers and Operations Research, 34*, 3126–3142.

Jolai, F., Rabbani, M., Amalnick, S., Dabaghi, A., Dehghan, M., & YazdanParas, M. (2007). Genetic algorithm for bi-criteria single machine scheduling problem of minimizing maximum earliness and number of tardy jobs. *Applied Mathematics and Computation, 194*, 552–560.

Kim, Y. D., & Yano, C. A. (1994). Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics, 41*, 913–933.

Koksalan, M., & BurakKeha, A. (2003). Using genetic algorithms for single machine bi-criteria scheduling problems. *European Journal of Operations Research, 145*, 543–556.

MahdaviMazdeh, M., Sarhadi, M., & Hindi, K. S. (2007). A branch-and-bound algorithm for single-machine scheduling with batch delivery minimizing flow times and delivery costs. *European Journal of Operational Research, 183*, 74–86.

MahdaviMazdeh, M., Sarhadi, M., & Hindi, K. S. (2008). A branch-and-bound algorithm for single-machine scheduling with batch delivery and job release times. *Computers & Operations Research, 35*, 1099–1111.

Mason, A. J., & Anderson, E. J. (1991). Minimizing flow time on a single-machine with job classes and setup times. *Naval Research Logistics, 38*, 333–350.

Pinedo, M. (1995). *Scheduling: Theory, algorithms and systems*. New Jersey: Prentice-Hall.

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research, 120*, 228–249.

Schaller, J. (2002). Minimizing the sum of squares lateness on a single machine. *European Journal of Operational Research, 143*, 64–79.

Schaller, J. (2007). A comparison of lower bounds for the single-machine early/tardy problem. *Computers & Operations Research, 34*, 2279–2292.

Sen, T., Dileepan, P., & Lind, M. R. (1995). Minimizing a weighted quadratic function of job lateness in the single machine system. *International Journal of Production Economics, 42*, 237–243.

Su, L. H., & Chang, P. C. (1998). A heuristic to minimize a quadratic function of job lateness on a single machine. *International Journal of Production Economics, 55*, 169–175.

Supithak, W., Liman, S. D., & Montes, E. J. (2010). Lot-sizing and scheduling problem with earliness–tardiness and setup penalties. *Computers & Industrial Engineering, 58*, 363–372.

Suriyaarachchi, R. H., & Wirth, A. (2004). Earliness/tardiness scheduling with a common due date and family setups. *Computers & Industrial Engineering, 47*, 275–288.

Syswerda, G. (1989). *Uniform crossover in genetic algorithms. Proc. 3rd Int'l conference on genetic algorithms*. Massachusetts: Morgan Kaufmann Publishing.

Valente, J. M. S., & Gonçalves, J. F. (2009). A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers & Operations Research, 36*, 2707–2715.

Ying, K. C. (2008). Minimizing earliness–tardiness penalties for common due date single-machine scheduling problems by a recovering beam search algorithm. *Computers & Industrial Engineering, 55*, 494–502.

Web reference

Obitko, M. (1998), *Introduction to genetic algorithms*. <http://www.obitko.com/tutorials/genetic-algorithms> Accessed December 2010.