



Scheduling of loading and unloading operations in a multi stations transshipment terminal with release date and inventory constraints



Atiyeh Bazgosha, Mohammad Ranjbar*, Negin Jamili

Department of Industrial Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

ARTICLE INFO

Article history:

Received 28 October 2016

Received in revised form 18 January 2017

Accepted 23 January 2017

Available online 26 January 2017

Keywords:

Transshipment terminal

Parallel machine scheduling problem

Metaheuristic algorithms

ABSTRACT

In this paper, we study a transshipment scheduling problem with multiple identical loading/unloading stations and release date and inventory constraints. This problem is similar to the parallel machine scheduling problem where the makespan is to be minimized, which is an NP-hard problem in the scheduling theory. We formulate the problem as an integer linear programming model, which is solvable only for small-size instances by CPLEX solver in reasonable times. Also, we develop two constructive heuristic solution approaches, namely parallel and serial schedule generation schemes. We also develop three metaheuristic methods based on genetic algorithm, particle swarm optimization and cuckoo optimization algorithm. The developed solution methods have been compared using computational studies based upon 870 randomly generated test instances. The experimental results show that the parallel schedule generation scheme outperforms the serial one and the cuckoo optimization algorithm shows the best performance among the developed metaheuristic methods.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

In this article, we consider a cross docking system containing multiple dock doors, which can process one inbound or outbound truck at a time, which is an extension of Briskorn, Choi, Lee, Leung, and Pinedo (2010). It is assumed that the given set of dock doors are equipped similarly and have identical factors for loading and unloading activities, such as capacity and processing speed. Moreover, only one type of product is considered and the loading and unloading operations are distinguished by the inventory modification they made. In this regard, we separate the operations into two classes, positive and negative, so that positive operations represent unloading activities which increase the inventory level, and the negative ones account for loading activities that lead to a decrease in the inventory level. Each loading/unloading operation is executed by a specific vehicle, which has arrived at the terminal at a certain point in the time, named release date, and requires a predetermined processing time. Also, preemption of operations is not allowed, i.e. once docked, a truck must be fully loaded or unloaded before its departure. Furthermore, a limited storage space is also

supposed to exist inside the transshipment terminal where an initial inventory is held. Additionally, we assume that the inventory level is immediately decreased by the time a loading operation starts, whereas the modification made by an unloading operation is applied by the time it is completed.

Regarding the aforementioned parameters to be integers, this problem can be viewed as a parallel machine scheduling problem with inventory constraints. In other words, dock doors are supposed to act as production machines and loading/unloading operations are considered as the jobs to be processed. Since the objective is to minimize the makespan, according to the three field notation introduced by Graham, Lawler, Lenstra, and Kan (1979) this problem can be represented as $Pm|r_j, inv|C_{max}$.

Due to the increasing amount of attention paid to cross docking scheduling in recent years, there is an extensive literature dealing with this research area. Boysen and Fliedner (2010) and Van Belle, Valckenaers, and Cattrysse (2012) present an extensive overview in this direction and provide recent surveys of the scheduling systems in cross docking platforms. Briskorn et al. (2010) focus on single machine scheduling subject to inventory constraints, where all jobs are available at the beginning of the time horizon and they either increase or decrease the inventory level according to their type, so that it remains nonnegative at each time. The authors relax the capacity restrictions and determine the computational complexity of the problem for several cases with different objectives

* Corresponding author at: Department of Industrial Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran.

E-mail addresses: a.bazgosha@stu.um.ac.ir (A. Bazgosha), m_ranjbar@um.ac.ir (M. Ranjbar), negin.jamili@stu.um.ac.ir (N. Jamili).

such as minimization of total weighted completion time and the number of tardy jobs, and prove the strongly NP-hardness of the general versions. The problem of scheduling the trucks in a cross dock with a single dock door is also considered in [Vahdani and Zandieh \(2010\)](#). The authors apply five metaheuristic algorithms for such a problem to minimize the total operation times in the facility. In [Briskorn, Jaehn, and Pesch \(2013\)](#), the authors develop exact methods for tackling the single machine problem subject to inventory constraints, where the objective is to minimize the total weighted completion time. They also assume that all jobs are available at the beginning and the inventory's capacity is unlimited. [Briskorn and Leung \(2013\)](#) consider the similar problem to find a schedule such that the maximum lateness among all jobs is minimized, and present four branch and bound algorithms.

Introducing a basic model for scheduling trucks at cross docking terminals with multiple dock doors, [Boysen, Fliedner, and Scholl \(2010\)](#) assume that the terminal has two gates and model it as a two machine scheduling problem. They also show that minimizing the makespan is strongly NP-hard even if all processing times are equal. [Madani-Isfahani, Tavakkoli-Moghaddam, and Naderi \(2014\)](#) discusses about a truck scheduling problem in a multiple cross docks, where two types of delay times are considered and the objective is to minimize the total operation time or maximize the throughput of the cross docking system. In the problem proposed by [Alpan, Bauchau, Larbi, and Penz \(2008\)](#), cross docking with multiple doors and temporary storage is considered. To determine the optimal truck sequence such that the total cost is minimized, the authors develop a bounded dynamic programming. [Alpan, Ladier, Larbi, and Penz \(2011a\)](#) and [Alpan, Larbi, and Penz \(2011b\)](#) also study a cross dock scheduling problem for serving outbound trucks at multiple stack doors. They assume that the arrival sequence of inbound trailers is fixed and consider a First-Come-First-Served (FCFS) policy to assign the order of incoming trucks.

The contributions of this article are threefold: (1) we introduce $Pm|r_j, inv|C_{max}$ and formulate it as a linear integer programming model; (2) we develop two schedule generation schemes named as parallel and serial to create feasible solutions for the problem; and (3) we develop three metaheuristic algorithms to improve the created solutions by the parallel and serial schedule generation schemes.

The remainder of this paper is organized as follows. Section 2 deals with modeling the problem as a linear integer programming and solution methods are sketched in Section 3. Section 4 is devoted to the computational study and evaluation of the developed algorithms. Finally, conclusions and future research directions are presented in Section 5.

2. Problem statement and modeling

We assume that the transshipment terminal has a set M of $|M|$ identical docks, considered as identical parallel production machines. It is also assumed that one product type is handled. Set of loading and unloading operations is denoted by J , considered as the set of jobs in the corresponding parallel machine scheduling problem. This set is separated into two subsets, J^+ and J^- , so that J^+ consists of unloading operations (positive jobs) and J^- accounts for loading ones (negative jobs). Each job $j = 1, \dots, |J|$ has a processing time p_j and is processed with a single truck that arrives at a specific release date shown by r_j . Considering a storage space with a given capacity IC , it is assumed that its initial inventory level is I^{ini} , and δ_j defines the inventory modification made when job $j = 1, \dots, |J|$ is processed. This parameter takes a positive value for unloading operations and a negative amount for the loading ones. It should be mentioned that any reduction in inventory level is applied

immediately by starting a loading operation and the increase made by an unloading operation is done when it is completely processed. Clearly, inbound and outbound shipments quantity must be taken into account so that the inventory level remains nonnegative and does not exceed its capacity at each point of time.

[Table 1](#) summarizes the parameters and decision variables required to formulate the problem as a mathematical model.

Regarding the above notations, the formulation of this problem reads as follows.

$$\text{Min } C_{max} \quad (1)$$

subject to

$$C_{max} \geq \sum_{t=1}^{|T|} \sum_{i=1}^{|M|} tX_{jit}; \quad \forall j \in J \quad (2)$$

$$\sum_{t=1}^{|T|} \sum_{i=1}^{|M|} X_{jit} = 1; \quad \forall j \in J \quad (3)$$

$$\sum_{\tau=t-p_j+1}^t \sum_{i \in J^+ \setminus \{j\}} X_{jit} \leq B(1 - X_{jit}); \quad \forall i \in M; \quad \forall j \in J; \quad \forall t \in T \quad (4)$$

$$\sum_{j=1}^{|J|} \sum_{\tau=\max\{t+1, r_j+p_j\}}^{\min\{|T|, t+p_j\}} \sum_{i=1}^{|M|} X_{jit} \leq |M|; \quad \forall t \in T \quad (5)$$

$$\sum_{t=1}^{|T|} \sum_{i=1}^{|M|} tX_{jit} - p_j \geq r_j; \quad \forall j \in J \quad (6)$$

$$I_t = I_{t-1} + \sum_{j \in J^+} \sum_{i=1}^{|M|} \delta_j X_{jit} + \sum_{j \in J^-} \sum_{i=1}^{|M|} \delta_j X_{j,i,t+p_j}; \quad \forall t \in \{1, \dots, |T|\} \quad (7)$$

$$I_0 = I^{ini} + \sum_{j \in J^-} \sum_{i=1}^{|M|} \delta_j X_{j,i,p_j}; \quad (8)$$

$$I_t \leq IC; \quad \forall t \in T \quad (9)$$

$$X_{jit} \in \{0, 1\}; \quad \forall j \in J; \quad \forall t \in T \quad (10)$$

$$I_t \in \mathbb{Z}^+; \quad \forall t \in T \quad (11)$$

The objective function (1) minimizes the makespan, which is set greater than or equal to the completion time of the last job in

Table 1

Description of parameters and variables.

Parameters	Definitions
$M : \{1, 2, \dots, M \}$	Set of machines with index i
$ M $	Total number of machines
$J : \{1, 2, \dots, J \}$	Set of jobs with index j
$ J $	Total number of jobs
$T : \{0, \dots, T \}$	Set of times with index t
$ T $	An upper bound for the completion time of all jobs
IC	Capacity of storage space
I^{ini}	Initial inventory level
p_j	Processing time of job j
r_j	Release date of job j
Variables	Definitions
X_{jit}	Binary variable that takes the value of 1 if processing of job j is completed on machine i at time instant t and takes 0, otherwise
I_t	Inventory level of transshipment terminal at time instant t

constraint (2). Constraint (3) guarantees that processing of each job is completed on a single machine and in exactly one time instant. Constraint (4), in which B indicates a sufficiently large positive value, enforces the jobs to be executed with no preemption. Constraint (5) ensures that no more than $|M|$ jobs can be processed as parallel. Also, Constraint (6) shows that no job can be started to process before its release date. Constraints (7) and (8) are related to the inventory level and denotes the amount of inventory at time $t \in T \setminus \{0\}$ and $t = 0$, respectively, based on modifications made by each operation. Restriction imposed to storage capacity is stipulated in constraint (9). Finally, the last two constraints describe the type of the variables, in which \mathbb{Z}^+ is the set of non-negative integers.

3. Constructive heuristics

As presented in Briskorn and Pesch (2013), the mentioned problem is proven to be strongly NP-hard even in the case of a single machine and considering no release dates for jobs. Therefore, solving the developed mathematical model is highly time consuming and only applicable for small instances. In this section, we aim at developing two constructive heuristics. These heuristic algorithms are schedule generation schemes, named parallel and serial, to provide feasible schedules for the problem. To do so, we follow the basic idea of Kolisch (1996), where so-called parallel and serial scheduling method was extended for the classical resource-constrained project scheduling problem.

3.1. Parallel schedule generation scheme (PSGS)

The PSGS iterates over the time horizon of the problem and does time incrementation. This scheme consists of at most $|J|$ decision points, in each of which a set of eligible jobs is scheduled based on their priority. A priority list of jobs is given as an input to the PSGS. Similar to the activity-list representation in project scheduling (see Debels, De Reyck, Leus, & Vanhoucke, 2006), Fig. 1 depicts a priority list of jobs where jobs 5 and 4 has the highest and the lowest priorities among jobs, respectively.

The PSGS starts at time instant $t = 0$ and the time pointer is increased as the eligible jobs are scheduled. In addition to the previously mentioned notation, the required parameters are defined in Table 2 and the steps of this method are demonstrated in Algorithm 1.

5	3	1	2	4
---	---	---	---	---

Fig. 1. Priority list of jobs.

Table 2
The required notation for PSGS.

Parameters	Definitions
PL	Priority list of jobs
$EJ_t : \{j \in J r_j \leq t\}$	Set of eligible jobs at decision point t (with release date not less than time t), sorted according to PL
AJ_t	Set of jobs in progress at time t , also called <i>active jobs</i>
ft_j	Finish time of job j
$EJ_t(g)$	g th member of EJ_t
AM_t	Number of available machines at time t
φ_t	Number of jobs that are finished at time t

Algorithm 1: Pseudo-code for the PSGS

Step 1. Let $\tau = \min_{j \in J} \{r_j\}$, $g = 1$, $AJ_\tau = \emptyset$,
 $|T| = \max_{j \in J} \{r_j\} + \sum_{j \in J} p_j$,
for all $t \in T$ let $I_t = I^{ini}$ and $AM_t = |M|$, for all $j \in J$ let $ft_j = 0$.

Step 2. Let $EJ_\tau = \{j \in J | r_j \leq \tau\}$ and $PL = PL \setminus EJ_\tau$.

Step 3. **for** $g = 1$ to $|EJ_\tau|$
if $\delta_{EJ_\tau(g)} < 0$ and for all $t \in \tau$: $I_t + \delta_{EJ_\tau(g)} \geq 0$
then
for all $t \geq \tau$ let $I_t = I_t + \delta_{EJ_\tau(g)}$.
for all $t \in [\tau, \tau + p_{EJ_\tau(g)}]$: let $AJ_t = AJ_t \cup EJ_\tau(g)$,
 $EJ_\tau = EJ_\tau \setminus EJ_\tau(g)$ and $AM_t = AM_t - 1$.
let $ft_g = \tau + p_{EJ_\tau(g)}$.
else if $\delta_{EJ_\tau(g)} > 0$ and for all $t \geq \tau + p_{EJ_\tau(g)}$:
 $I_t + \delta_{EJ_\tau(g)} < IC$ **then**
for all $t \geq \tau + p_{EJ_\tau(g)}$ let $I_t = I_t + \delta_{EJ_\tau(g)}$.
for all $t \in [\tau, \tau + p_{EJ_\tau(g)}]$: let $AJ_t = AJ_t \cup EJ_\tau(g)$,
 $EJ_\tau = EJ_\tau \setminus EJ_\tau(g)$ and $AM_t = AM_t - 1$.
let $ft_g = \tau + p_{EJ_\tau(g)}$.

Step 4. **if** $AM_\tau = 0$ **then**
let $\tau = \min_{j \in AJ_\tau} \{ft_j\}$, $AM_\tau = AM_\tau + \varphi_\tau$ and
 $AJ_\tau = AJ_\tau \setminus \{j \in AJ_\tau | ft_j = \tau\}$.
go to Step 2.

Step 5. **if** ($EJ_\tau = \emptyset$ and $PL \neq \emptyset$) **then**
let $\tau = \min_{j \in PL} \{r_j\}$; go to Step 2.
else if ($EJ_\tau = \emptyset$ and $PL = \emptyset$) **then**
let $C_{max} = \max(ft_1, \dots, ft_n)$.
return feasible schedule and stop.
else if ($EJ_\tau \neq \emptyset$ & ($PL \neq \emptyset$ or $AJ_\tau \neq \emptyset$))
let $\tau_1 = \min_{j \in PL} (r_j)$, $\tau_2 = \min_{j \in AJ_\tau} (ft_j)$ and
 $\tau = \min(\tau_1, \tau_2)$.
if $\tau = \tau_1$ **then** go to Step 2.
else if $\tau = \tau_2$ **then**
let $AM_\tau = AM_\tau + \varphi_\tau$, $AJ_\tau = AJ_\tau \setminus \{j \in AJ_\tau | ft_j = \tau\}$
and go to Step 3.
else if ($EJ_\tau \neq \emptyset$ and $PL \neq \emptyset$ and $AJ_\tau \neq \emptyset$) no feasible solution can be found, stop.

The above algorithm initializes the required parameters in Step 1, where the main one is τ which defines the time pointer and is set to $\min_{j \in J} \{r_j\}$ at the beginning of the procedure. In the PSGS, τ is increased gradually while a set of eligible jobs are scheduled according to the priority list of PL . Thereafter, based on the value of τ , PL and EJ_τ are updated in Step 2. It is to be noted that in each stage of updating τ , some elements are removed from PL and put into EJ_τ in which the scheduled jobs are no longer maintained.

In Step 3, the possibility of scheduling is checked for all the eligible jobs in EJ_τ . Since the elements of EJ_τ are regarded as the eligible jobs which are available at the current time, they are all needed to be investigated in order not to violate the inventory constraints in the case of being scheduled. Checking procedure is separated in this step for the positive and negative set of jobs, as the modification they made in inventory level is at the start of the processing for the negative ones and at the end of it for the positives. The inventory level of the storage space (I_t), the set of active jobs (AJ_τ), the number of available machines (AM_τ) and the set of eligible job (EJ_τ) are parameters to be updated at every

Table 3
Parameters of the example.

j	1	2	3	4	5
p_j	6	4	6	2	7
r_j	2	4	0	5	3
δ_j	7	6	-3	-5	-11

repetition of Step 3 where a job is newly scheduled. The finish time of the mentioned job is also calculated in this step. Afterwards, there are two possible cases which are investigated in the following. Step 4 is to consider a situation in which no idle machine is available to start an operation. In this case, the time pointer (τ) is increased to the minimum of the finish times of the active jobs and Step 2 is repeated. Step 5 is then designed to examine the situation where no eligible job is in EJ_τ at time τ , while set PL contains some jobs yet. To schedule these jobs, τ is set to $\min_{j \in PL}\{r_j\}$ so that the

mentioned jobs are added to EJ_τ , the algorithm is then restarted from Step 2. In Step 5, it is also investigated that if all the jobs are scheduled, the objective function is calculated using $C_{max} = \max\{ft_1, \dots, ft_n\}$ and the procedure is terminated. Another possibility is that some jobs in EJ_τ are not scheduled because of the inventory restriction, while PL or AJ_τ are not empty. To tackle this situation, the time pointer (τ) can be increased to $\tau_1 = \min\{r_j\}$, in order to add some jobs to EJ_τ , or to the minimum of the finish times of the active positive jobs (τ_2) when the inventory level is changed. It is worth mentioning that completion of the negative job is not considered since the modification they make to the inventory level is applied at the start of their processing. The earliest change made in the inventory level is computed by considering the minimum of τ_1 and τ_2 and the time pointer is updated to this value. Another case, leading to the failure of this method in finding a feasible solution, is a situation in which PL and AJ_τ are empty, while the eligible jobs in EJ_τ are not permissible to be processed due to the inventory constraints. In this case no modification can be made and the procedure stops searching for a feasible solution.

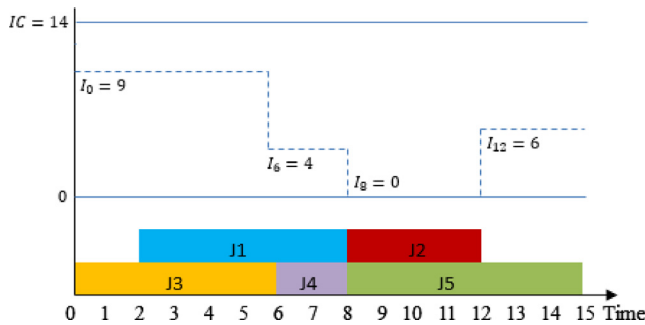


Fig. 2. Gantt Chart for the optimal schedule of the example.

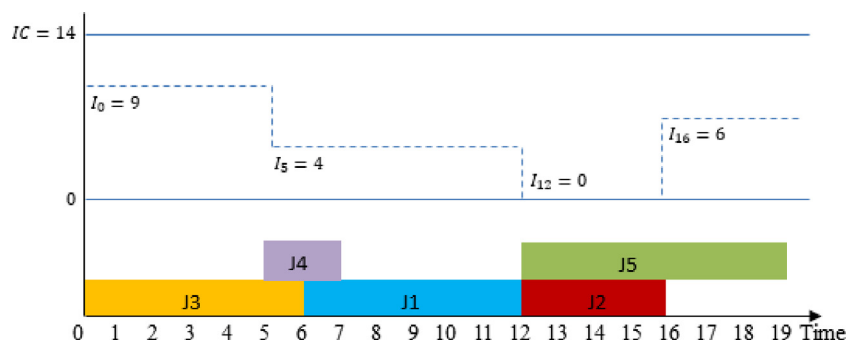


Fig. 3. Gantt Chart of the example obtained by the PSGS.

Consider an instance where due to the inventory constraints the set of positive and negative jobs are not possible to be scheduled separately. In such a case, a feasible solution may be found by scheduling a negative and positive job simultaneously, such that the processing of the negative job starts right at the completion time of the positive one. As performed in Step 6, in these cases the PSGS fails to obtain a feasible solution.

In Step 7, the completion time of the scheduled job, the availability of machines and the three sets AJ_τ , SJ and EJ_τ are updated, and in the case of existence of an unscheduled job the procedure is repeated from Step 5.

Consider an example of $Pm|r_j, inv|C_{max}$ with $|J| = 5$, $|M| = 2$, $I^{ini} = 12$, $IC = 14$, $PL = (5, 3, 1, 4, 2)$. Other required parameters of this example is provided in Table 3.

To employ the PSGS, we do as follows. At the beginning, the only eligible job to be started at time 0 is job 3. Having been completed, this job increases the time pointer to $\tau = 2$, where job 1 is the single job able to be started based on its release date. As the inventory constraints is violated by assigning this job, the time pointer changes to $\tau = 3$ and the list of eligible sorted jobs is defined as $EJ_\tau = (5, 1)$. Regarding the release dates, since the eligible jobs are not allowed to be processed, the decision point is modified and takes the value of $\tau = 4$. Iterating Step 2, we update the eligible list to $EJ_\tau = (5, 1, 4, 2)$, where no eligible job is available to be processed, and the decision point changes to $\tau = 5$. According to the new list $EJ_\tau = (5, 1, 4, 2)$, job 4 is the only job permissible to be processed. Scheduling the mentioned job leads to the non-accessibility of the machine and increment the time pointer by one. Thereafter, job 1 is chosen among the eligible list to be started, and the updated decision point is equal to 12 (makespan of the currently scheduled jobs). Finally, the processing of job 5 and 2 begins simultaneously at this time point and a feasible schedule with $C_{max} = 19$ is obtained.

In the following, Fig. 2 depicts a Gantt Chart for the optimal schedule, while the one based on the obtained schedule from PSGS is shown in Fig. 3.

As illustrated before, in some instances the PSGS is unable to reach a feasible solution for a specific priority list. Let us consider the previously mentioned example where $PL = (5, 3, 2, 1, 4)$. Fig. 4 depicts a partial schedule for this problem where the only eligible job at time $\tau = 0$ is job 3. Thereafter, at time $\tau = r_1 = 2$ job 1 is eligible to be processed while it is not permissible because of the inventory restriction. τ is then increased to 3 ($\tau = r_5 = 3$) and EJ_3 is updated to $(5, 1)$. Since none of the jobs in EJ_3 are permissible to be scheduled, τ is updated to 4 ($\tau = r_2 = 4$) and $EJ_4 = (5, 2, 1)$ and none of these jobs can be processed yet. In the next step, $\tau = r_4 = 5$, $EJ_5 = (5, 2, 1, 4)$ and job 4 is chosen to be processed. In the following where $\tau = ft_3 = 6$, $EJ_6 = (5, 2, 1)$ and job 2 can be started in this time. Next, the decision point is updated as $\tau = ft_4 = 7$ and $EJ_7 = (5, 1)$ but none of these two jobs is allowed to be started in this time. The final decision point is $\tau = ft_2 = 10$

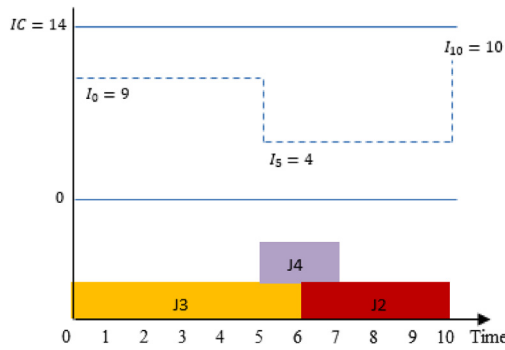


Fig. 4. Gantt Chart for the partial infeasible schedule of the example obtained by PSGS.

and there is situation like previous decision point in which remaining jobs (1 and 5) are not admissible for scheduling and no feasible solution is found.

Table 4

The required notation for SSGS.

Parameters	Definitions
RE	An ordered set of jobs that are not scheduled in the first run of Step 2
$PL(g)$	gth member of OJ
ns	A boolean variable

3.2. Serial schedule generation scheme (SSGS)

The approach of the SSGS is to schedule each job sequentially and as soon as possible with regard to the inventory constraints. Similar to PSGS, the SSGS gets a priority list PL as input to construct a schedule. In addition to previously introduced notations, we use the notation introduced in Table 4 to describe this method as given in Algorithm 2.

Algorithm 2: Pseudo-code for the SSGS

```

Step 1. Let  $RE = \emptyset$ ,  $|T| = \max\{r_j\} + \sum_{j \in J} p_j$ ,  $ns = FALSE$ ,
for all  $t \in T$  let  $I_t = I^{ini}$  and  $AM_t = |M|$ , for all  $j \in J$  let  $ft_j = 0$ .

Step 2. for  $g = 1$  to  $|J|$ 
    let  $\tau = \infty$ ;
    if  $\delta_{PL(g)} < 0$  then
        let  $\tau = \min\{t : t \geq r_{PL(g)}, I_t + \delta_{PL(g)} \geq 0, AM_{t'} > 0 : \forall t' = t, \dots, t + p_{PL(g)}\}$ .
        if  $\tau < \infty$  then
            for all  $t \geq \tau$  let  $I_t = I_t + \delta_{PL(g)}$ .
            for all  $t \in [\tau, \tau + p_{PL(g)}]$ : let  $AM_t = AM_t - 1$ .
            let  $ft_{PL(g)} = \tau + p_{PL(g)}$ ,  $ns = TRUE$ .
        else  $RE = RE \cup PL(g)$ .
    else if  $\delta_{PL(g)} > 0$  then
        let  $\tau = \min\{t : t \geq r_{PL(g)}, I_{t+p_{PL(g)}} + \delta_{PL(g)} \leq IC, AM_{t'} > 0 : \forall t' = t, \dots, t + p_{PL(g)}\}$ .
        if  $\tau < \infty$  then
            for all  $t \geq \tau + p_{PL(g)}$  let  $I_t = I_t + \delta_{PL(g)}$ .
            for all  $t \in [\tau, \tau + p_{PL(g)}]$ : let  $AM_t = AM_t - 1$ .
            let  $ft_{PL(g)} = \tau + p_{PL(g)}$ ,  $ns = TRUE$ .
        else  $RE = RE \cup PL(g)$ .

Step 3. if  $|RE| = 0$  then
    let  $C_{max} = \max(ft_1, \dots, ft_n)$ .
    return feasible schedule and stop.
else if ( $ns = FALSE$ )
    let  $J = RE$ ,  $RE = \emptyset$ ,  $ns = FALSE$ .
    go to Step 2.
else no feasible solution can be found, stop.

```

In the SSGS, the first step is to initialize the main parameters. In Step 2, which is regarded as the most important stage, the jobs are selected for scheduling based on priority list PL . To do so, τ is considered to define the appropriate time for starting the process of each job which is computed with regard to its release date, availability of the machines and the inventory level of storage space. This step is separated into two parts to distinguish the modifications made by the positive and negative jobs. Considering a specific job, if the value of τ is limited, the job is scheduled and the associated parameters such as its finish time, the inventory level and the number of idle machines are updated. Otherwise, the given job is put into RE set to be considered to

scheduling in the next round. If no unscheduled job remains, C_{max} is calculated in Step 3. Otherwise, the jobs in set RE are investigated to be scheduled. In this regard, we place the elements of RE in J , let $RE = \emptyset$ and go to Step 2. In this stage, various cases may happen. If all the remaining jobs are scheduled, the procedure returns a feasible schedule and stops. In some situations where there are still some unscheduled jobs remained after the repetition of Step 2, this step is iteratively repeated to reach a feasible solution. It is to be mentioned that if this step leads to no further changes in the partial schedule, ns is set to $FALSE$ and the algorithm is unable to find a feasible schedule according to the given priority list.

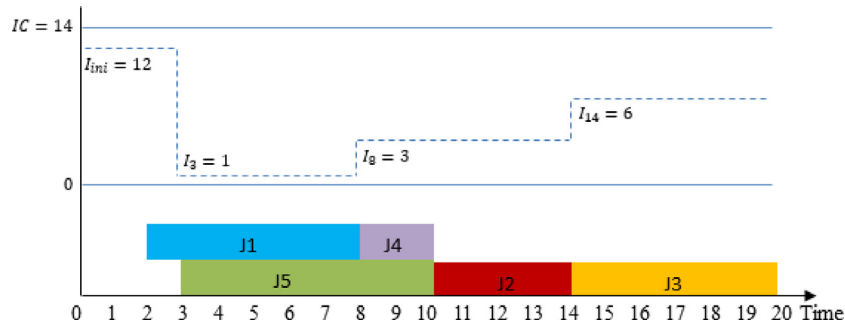


Fig. 5. Gantt Chart of the example obtained by the SSGS.

According to the priority list $PL = (5, 3, 1, 4, 2)$, job 5 is the first job to be started at time $\tau = 3$. As job 3, which is chosen as the following job, is not eligible to be started due to the inventory constraints, this job is added to RE . Thereafter, job 1 starts its processing at time $\tau = 2$. Subsequently, the processing of job 4 begins at time $\tau = 8$. Then job 2 starts at time $\tau = 10$ and the only member of set RE (job 3) is then available for processing. Thus, job 3 is scheduled to be started at $\tau = 14$ and a feasible schedule with $C_{max} = 20$, shown in Fig. 5, is obtained.

In the aforementioned example, if the priority list is defined as $PL = (4, 3, 2, 5, 1)$, the processing of job 4 begins at $\tau = 5$. Thereafter, jobs 3 and 2 are scheduled at $\tau = 0$ and $\tau = 6$, respectively. According to this partial scheduling, depicted in Fig. 6, the remaining jobs (1 and 5) are not permissible to be processed due to the inventory constraints and there is no feasible schedule for this instance.

4. Metaheuristic approaches

In this section we develop three metaheuristic approaches based on the genetic algorithm (GA), the particle swarm optimization (PSO) and the cuckoo optimization algorithm (COA). These methods are provided to improve the solution generated by the PSGS and SSGS. Based on the ideas developed for the solution approaches of the well-known resource-constrained project scheduling problem such as Ranjbar, De Reyck, and Kianfar (2009) and Ranjbar and Kianfar (2007), each of these three algorithms is combined with either PSGS or SSGS to work. In other words, our developed metaheuristics generate priority lists where each priority list is transformed to a schedule using one of the developed schedule generation schemes.

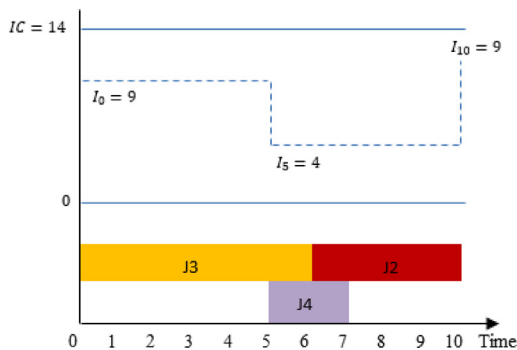


Fig. 6. Gantt Chart for the partial infeasible schedule of the example obtained by SSGS.

4.1. Genetic algorithm

A genetic algorithm is a population-based metaheuristic optimization method inspired by biological evolution, which is invented by Holland (1975). To optimize a problem, a population is constructed in which each solution's quality is evaluated by a predefined "fitness function". The algorithm repeatedly modifies the population by forming new generations and explores the search space to move into a favorable direction.

In the problem $Pm|r_j, inv|C_{max}$, a solution (chromosome) is represented as a priority list of jobs and a population is constructed by a set of such lists.

Algorithm 3 demonstrates the framework of the GA in several steps.

Algorithm 3: Pseudo-code for the GA

- Step 1.** Initialize population Pop by randomly generating $|Pop|$ solutions.
- Step 2.** for all $i \in Pop$ calculate C_{max} using PSGS or SSGS.
- Step 3.** Let $i = 1$.
- Step 4.** Consider the i th member of Pop as the first parent and select its partner, namely $j \in Pop$ where $j \neq i$, via Roulette wheel selection strategy.
- Step 5.** Perform two point crossover operator over the selected parents to generate a new child solution.
- Step 6.** Apply the mutation operator over the new generated child solution with the probability of P_{mut} and add it to the new population ($NewPop$).
- Step 7.** Let $i = i + 1$.
 if $i \leq |Pop|$ then
 go to Step 4.
 else if the stopping criterion has no met then
 replace the worst solution of $NewPop$ with the best solution of Pop .
 let $Pop = NewPop$, $NewPop = \emptyset$ and go to Step 2.
 else Stop.

The developed algorithm is a usual version of GA and includes 7 steps. In the first step, an initial population is constructed, consisting of $|Pop|$ randomly generated solutions where each one is a priority list. In Step 4, two parent solutions are selected to pair using roulette wheel strategy. The main idea of the roulette wheel is to associate more chance to better individuals, proportional to their fitness, to be selected. To do so, in the first step a probability is assigned to each of the chromosomes to define their chance to be transferred to the pool of solutions. The probability of chromosome i is calculated using $\frac{1/C_{max_i}}{\sum_i (1/C_{max_i})}$ and each chromosome is considered

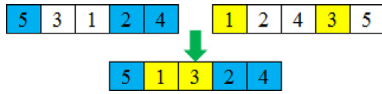


Fig. 7. An example of two point crossover.

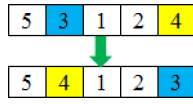


Fig. 8. An example of mutation operator.

as a part of the roulette wheel. In Step 5, a two point crossover operator is applied on the parents and a new child solution is generated. Fig. 7 illustrates an example of this operator, whereas cells 1 and 4 are the two crossing points.

Mutation operation is performed in Step 6. This operator is done on the newly generated child solution with the probability of P_{mut} , such that two random cells of a solution are selected and exchanged in order to produce a new chromosome. This process is shown in Fig. 8, where the solution is mutated by shifting cells 2 and 5.

4.2. Particle swarm optimization

Particle swarm optimization is another population-based search algorithm, designed and introduced by Eberhart and Kennedy (1995), originated from schooling behavior of fish or flocking behavior of birds. This algorithm was originally intended for simulating social behavior of birds or fish, randomly searching food in an area. A PSO algorithm contains a swarm of particles flying through a search space where their position is adjusted according to their own experience and the experience of their neighbors. Hence, the movement of the particles is guided by the entire swarm's best known position. Considering the particles as the candidate solutions to the problem, in PSO particles move toward optima by updating the knowledge gained by the entire swarm. The features of each particle are their velocity and position, such that a particle's position is modified based on its velocity, and the velocity is determined by regarding both local and global best information and defined by the following elements.

- The personal best solution (the best position particle k has visited, denoted by $Pbest_k$).
- The global best (the best position visited by the entire swarm, demonstrated by $Gbest$).

4.2.1. Solution representation

Representation of a solution is a key issue in any optimization problem and has a great influence on the algorithm performance.

0	0	0	0	1
0	0	1	0	0
1	0	0	0	0
0	1	0	0	0
0	0	0	1	0

Fig. 9. Position matrix of a particle.

In PSO, we use two types of solution representations. The first one is similar to the solution representation proposed for the GA approach, depicted in Fig. 1, in which a list of prioritized jobs is considered where the sequence of jobs shows their priority. In the second representation, see Fig. 9, a solution is illustrated as a $|J| \times |J|$ binary matrix, namely position matrix, in which each column accounts for a job and each row represents the priorities, whereas only one element has the value of 1 in each column. Assume that (i, j) is the cell associated with the i th row and j th column in the k th particle's position matrix. If this element is equal to 1 then i th priority is assigned to the j th job among all jobs. Fig. 9 corresponds to the priority list shown in Fig. 1.

4.2.2. Velocity and position of particles

Velocity of each particle is determined based on both local and global best information and presented as a $|J| \times |J|$ matrix whose elements are in range $[-V_{max}, V_{max}]$. At the beginning of the algorithm, this matrix is initialized randomly so that the velocities will be uniformly disturbed in the interval $[-V_{max}, V_{max}]$. Thereafter, the velocity matrix is updated in each iteration using Eq. (12), introduced in Izakian, Ladani, Abraham, and Snasel (2010), where the local and global best positions are employed as well as the current position of the given particle. The position matrix is then obtained according to Eq. (13). The required parameters for the PSO approach is summarized in Table 5.

$$V_k^{itr+1}(i, j) = V_k^{itr}(i, j) + c_1 r_1 (X_{Pbest_k}^{itr}(i, j) - X_k^{itr}(i, j)) + c_2 r_2 (X_{Gbest}^{itr}(i, j) - X_k^{itr}(i, j)) \quad (12)$$

$$X_k^{itr}(i, j) = \begin{cases} 1 & \text{if } V_k^{itr}(i, j) = \max\{V_k^{itr}(i, j)\}; \quad \forall i \in J \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

In Eq. (12), c_1 and c_2 are considered as the positive constants to determine whether the local position or the global one has the priority rather than the other. Moreover, r_1 and r_2 are chosen randomly from the interval $[0, 1]$.

4.2.3. The PSO framework for the problem $Pm|r_j, inv|C_{max}$

The scheme of the PSO method in this paper, consisting of two phases, is sketched as presented in Algorithm 4.

Table 5

The required notation for PSO.

Parameters	Definitions
X_k^{itr}	The position matrix of k th particle in itr th iteration
$X_{Pbest_k}^{itr}$	The position matrix of $Pbest_k$ in itr th iteration
X_{Gbest}^{itr}	The position matrix of $Gbest$ in itr th iteration
V_k^{itr}	The velocity matrix of k th particle in itr th iteration
ER	Sum of eligible rows in the velocity matrix
c_1, c_2	Positive constants by which the influence or priority of local and global positions is controlled
r_1, r_2	Random values in range $[0, 1]$

Algorithm 4: Pseudo-code for the PSO

Phase 1: Initialize a swarm by randomly generating $|Pop|$ particles

Step 1. Generate the elements of velocity matrix from a discrete uniform distribution in the interval $[-V_{max}, V_{max}]$ and set $ER = J$.

Step 2. Let $j = 1$.

Step 3. **if** $V_k^{itr}(i', j) = \max_{i \in ER} V_k^{itr}$ whereas $i' \in J$ **then**
 let $X_k^{itr}(i', j) = 1$.
 omit i' from ER and for all $i \in J \setminus \{i'\}$ let $X_k^{itr}(i, j) = 0$.
if $j < |J|$ **then**
 let $j = j + 1$ and go to Step 3.

Phase 2: PSO implementation

Step 4. Let $itr = 0$, for all $i, j \in [0, |J|]$ let $X_{Gbest}^{itr}(i, j) = X_0^{itr}(i, j)$.

Step 5. **for** all $i, j \in [0, |J|]$, $k \in [0, |Pop|]$ let $X_{Pbest_k}^{itr}(i, j) = X_k^{itr}(i, j)$.

if $C_{max}(X_k^{itr}) < C_{max}(X_{Gbest}^{itr})$ **then**
 for all $i, j \in [0, |J|]$ let $X_{Gbest}^{itr}(i, j) = X_k^{itr}(i, j)$.

Step 6. **for** all $i, j \in [0, |J|]$, $k \in [0, |Pop|]$ let
 $V_k^{itr+1}(i, j) = V_k^{itr}(i, j) + c_1 r_1 (X_{Pbest_k}^{itr}(i, j) - X_k^{itr}(i, j)) + c_2 r_2 (X_{Gbest}^{itr}(i, j) - X_k^{itr}(i, j))$.
 Let $j = 1$.

Step 7. **if** $V_k^{itr}(i', j) = \max_{i \in ER} V_k^{itr}(i, j)$ whereas $i' \in J$ **then**
 let $X_k^{itr}(i', j) = 1$.
 omit i' from ER and for all $i \in J \setminus \{i'\}$ let $X_k^{itr}(i, j) = 0$.
if $j < |J|$ **then**
 let $j = j + 1$ and go to Step 7.

Step 8. Let $itr = itr + 1$.
if $C_{max}(X_k^{itr}) < C_{max}(X_{Pbest_k}^{itr})$ **then**
 for all $i, j \in [0, |J|]$ let $X_{Pbest_k}^{itr}(i, j) = X_k^{itr}(i, j)$.
if $C_{max}(X_k^{itr}) < C_{max}(X_{Gbest}^{itr})$ **then**
 for all $i, j \in [0, |J|]$ let $X_{Gbest}^{itr}(i, j) = X_k^{itr}(i, j)$.

Step 9. **if** the stopping criterion has not met **then**
 let $ER = J$ and go to Step 8.
else Stop.

The developed algorithm of this section initializes the population by randomly generating the velocity matrix of particles and the particles' positions are determined according to the given equation in Step 3. In this step, in order to avoid assigning a priority to more than one job, having determined the maximum value of each column, we then omit the associated row from the set of eligible rows.

In the second phase, the position matrix of $Pbest_k$ and $Gbest$ for $k \in [0, |Pop|]$ is determined by the current information and the velocity matrix of particles is updated in Step 6, afterwards. Thereafter, the position matrix of particles, $Pbest_k$ and $Gbest$ are updated in Step 7 and 8, respectively. It is to be noted that C_{max} , defined in Steps 5 and 8, is calculated using one of the two developed schedule generation schemes.

4.3. Cuckoo optimization algorithm

Cuckoo optimization algorithm is one of the latest evolutionary algorithms developed by [Rajabioun \(2011\)](#). This population-based metaheuristic is inspired by the life of a bird family, called Cuckoo, and starts with an initial population of these birds. Based on the brood parasitism of these species, cuckoos lay their eggs in the habitats of other host birds. Whilst some of these eggs are detected and expelled out by the host birds, some have the opportunity to grow up. The number of survived eggs in each area indicates the nest suitability of that area. Therefore, the best environment in

which more eggs survive will be the term that COA is going to optimize. In this process, the remained eggs turn into mature cuckoos and make some groups with specific habitat regions. As the best habitat of all groups turns into a destination for cuckoos in other societies, they move toward this area and reside in some places near the best area. The number of eggs each cuckoo has and its distance to the best residence are calculated to consider some egg laying radius, and the cuckoos start to lay eggs in some random nests inside this radius. Iterating this process, the best position with the maximum profit, in where the most of cuckoo population is gathered, is obtained.

In this approach, k – mean clustering method is used for grouping the cuckoos. This partitioning process is the first and simplest of all clustering algorithms, introduced by [MacQueen \(1967\)](#), which is applicable to various problems. The procedure starts with partitioning a population into k sets. Computing the means of the points in each set, a new partition of points is formed and the points are put into other groups. This process continues in order to minimize the average square distance of each group's point from its mean. In the following, the parameters required for proposing the COA method are introduced in [Table 6](#).

In this method, by the best solution, we mean the center of a cluster with the minimum mean. Similar to the GA and PSO, a solution, representing a habitat that shows the current living position of a cuckoo, is defined by a priority list of activities and a randomly generated set of these solutions forms the initial population.

Table 6

The required notation for COA.

Parameters	Definitions
$NumE_i$	The number of i th cuckoo's eggs
$MinE_i$	Minimum number of eggs for i th cuckoo which is a constant number
$MaxE_i$	Maximum number of eggs for i th cuckoo which is a constant number
ELR_i	Egg laying radius for i th cuckoo
Pop_i^{itr}	The i th habitat in the itr th iteration
Pop_{best}^{itr}	The best habitat in the itr th iteration
CN	Total number of clusters
F	Movement coefficient, a random number between 0 and 1
MI_{ij}	The distance between habitat i and j
MI_{ibest}	Distance of habitat i to the best habitat

4.3.1. The COA framework for the problem $Pm|r_j, inv|C_{max}$

We outline the framework for the COA approach as given in Algorithm 5.

Algorithm 5: Pseudo-code for the COA

- Step 1.** Randomly generate $|Pop|$ cuckoo habitats as the initial population Pop .
- Step 2.** Dedicate some eggs to each cuckoo.
for all $i \in Pop$ let
 $NumE_i = (MaxE_i - MinE_i) * rand[0, 1] + MinE_i$.
- Step 3.** Calculate the egg laying radius for each cuckoo
 $i \in Pop$ as $ELR_i = \frac{NumE_i}{\sum_i NumE_i} * |J|$.
- Step 4.** Define the position of a cuckoo's eggs inside its corresponding radius.
- Step 5.** Evaluate the habitat of each egg by calculating C_{max} using PSGS or SSGS.
If the number of eggs is more than $|Pop|$ then
omit those who live in worse habitats.
- Step 6.** Cluster cuckoos using $k - mean$ method and find the center of the best habitat (the cluster with the minimum mean) as the best solution found.
- Step 7.** Move the new cuckoo population toward the best habitat.
- Step 8.** If stopping criterion (maximum number of iteration) is met then
Stop.
else go to Step 2.

Regarding the problem $Pm|r_j, inv|C_{max}$, the value of ELR_i , calculated in step 3, determines the maximum number of jobs allowed to be shifted in the priority list. In step 4, for all $i \in Pop$ generate $NumE_i$ random numbers, uniformly distributed in interval $[1, ELR_i]$. Next, for all $i \in Pop$ choose $NumE_i$ pairs of jobs and then exchange the position of each pair in the priority list.

To develop the clustering technique mentioned in Step 6, we randomly choose CN elements as the centers of clusters. In this paper, this parameter is computed using Eq. (14).

$$CN = \frac{\max_{i \in [0, |Pop|]} C_{max}(Pop_i^{itr}) - \min_{i \in [0, |Pop|]} C_{max}(Pop_i^{itr})}{4} + 1 \quad (14)$$

Afterwards, the distance of each member from the center of its cluster is determined. Considering MI_{ij} as the distance between i th member and center j , this parameter is represented by a triple (a, b, c) where a is a specific priority that is assigned to different jobs in i and j and b and c are the jobs associated with the a th priority in i and j , respectively. To elaborate more, assume i and j as depicted in

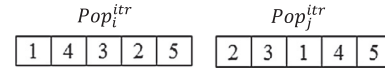
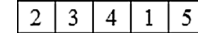
**Fig. 10.** Two example of habitats.**Fig. 11.** Pop_k^{itr+1} .

Fig. 10 in which MI_{ij} is then determined as $MI_{ij} = \{(1, 1, 2), (2, 4, 3), (3, 3, 1), (4, 2, 4)\}$.

Having calculated the distances, we then dedicate each member to a cluster where its center is nearest to the position of mentioned member in comparison with other clusters. Subsequently, the average of C_{max} is computed and the element with the nearest value to the average is considered as the related cluster's center. Evaluating the new distance, the process is performed repeatedly till no change in the clusters is occurred.

In Step 7, in order to move k th egg of the i th cuckoo (generating Pop_k^{itr+1}), we choose the first $[F * |MI_{ibest}|]$ elements of MI_{ibest} and apply the required changes on Pop_k^{itr} . In the following, an example is presented for further explanation.

Let us assume that Pop_{best}^{itr} is the j th member given in Fig. 10 and F is equal to 0.6. The first two elements of MI_{ibest} , $(1, 1, 2)$ and $(2, 4, 1)$, are chosen and Pop_k^{itr+1} is updated by replacing job 1 with job 2 and job 4 with job 1 in Pop_k^{itr} . Exchanging the jobs, Pop_k^{itr+1} is obtained as shown in Fig. 11.

5. Computational experiments

To investigate the performance of the developed solution approaches, we generated 870 random test instances in three groups, including small, medium and large size data. The algorithms were coded in C++ and ran on a portable computer with an Intel Core i3 and 4 GB of RAM, and Microsoft Windows 8.1 operating system. The integer programming model is also solved by the ILOG CPLEX 12.6. All test instances and detailed computational results can be found at http://www.m_ranjbar.profcmis.um.ac.ir/index.php/submitted-articles/11357.

5.1. Data set generation

We designed the test instances based upon the following parameters. The number of jobs ($|J|$) for small, medium and large size instances is taken from sets $\{5, 10, 15, 20\}$, $\{30, 35, 40\}$, and $\{50, 55, 60\}$, respectively, and the number of machines are randomly chosen from sets $\{1, 2\}$, $\{2, 3, 4\}$ and $\{2, 3, 4, 5\}$, sequentially. The processing time of each job (p_j) is uniformly distributed in the interval $[1, 10]$. Having generated the processing times, we then choose the release dates randomly from the discrete uniform distribution $U[0, 1/4 * \sum_i p_i]$. Moreover, the inventory modification made by each job (δ_j) is created randomly from integer values in $[1, 10]$. Consequently, the initial inventory level (I^{ini}) is an integer value drawn from $U[\max\{0, \sum_{j \in J^-} \delta_j\}, \sum_{j \in J^-} \delta_j]$ and the storage capacity (IC) follows a discrete uniform distribution in the interval $[I^{ini} + \max\{(\sum_{j \in J^+} \delta_j - \sum_{j \in J^-} \delta_j), 0\}, I^{ini} + \sum_{j \in J^+} \delta_j]$.

To determine the set of positive and negative jobs, $\alpha \in \{0.4, 0.5, 0.6\}$ is regarded as the percentage of positive ones. Finally, for each of these 87 combinations, we generated 10

Table 7

The run time of model using CPLEX.

Number of machines	Number of jobs				Average
	5	10	15	20	
1	0.196	0.688	7.899	266.648	68.857
2	0.263	1.257	12.555	358.886	93.240
Average	0.229	0.972	10.227	312.767	81.048

random instances, so that 870 test instances are considered when evaluating the developed solution methods.

5.2. Performance of the model

In this section, we investigate the performance of the developed integer linear programming model. Since the optimal results, obtained by ILOG CPLEX, are only achievable for all small size instances and some of medium size instances, the results of the three metaheuristic algorithms for the mentioned test sets are compared to the optimal solutions while the results of other instances are compared to the best found solution among all developed approaches.

In Table 7, the average CPU run time of the mathematical model is provided for the small size data. As expected, increasing the number of jobs or machines leads to larger run time of the model.

Since reaching the optimality using CPLEX is considerably time consuming for medium and large size instances, we imposed a limit of 1800 s and 3600 s on the CPU Run times for these data sets. As given in Table 8, for some instances, CPLEX is not able to find optimal solution, or even a feasible one, within the prescribed time limits. Tables 8 and 9 contain the number of optimal and feasible solutions obtained for each class of instances, in terms of the number of jobs and machines, where the first values indicate the total number instances solved optimally and the second ones reports the number of instances for which at least a feasible solution has been found. It is to be mentioned that there is 30 instances for each group of test sets categorized in a cell.

As is evident in the above tables, the computational complexity of the problem is increased with increasing the number of jobs, while it is not inversely dependent to the number of machines. Following the given information, we can easily observe that in case of

Table 8Number of optimal and feasible solutions with $TL = 1800$ s.

Number of machines	Medium			Large		
	Number of jobs			Number of jobs		
	30	35	40	50	55	60
2	7/30	1/22	13/30	0/3	0/1	0/0
3	20/30	12/30	8/28	2/10	0/6	0/2
4	28/30	28/30	30/30	17/23	12/18	9/15
5				28/28	26/29	26/26

Table 9Number of optimal and feasible solutions with $TL = 3600$ s.

Number of machines	Medium			Large		
	Number of jobs			Number of jobs		
	30	35	40	50	55	60
2	13/30	13/30	1/29	0/9	0/4	0/3
3	22/30	17/30	9/28	3/14	0/10	0/5
4	28/30	30/30	30/30	19/26	11/25	17/20
				29/30	28/30	28/30

Table 10

The setting of parameters of metaheuristic algorithms.

Algorithm	Parameter	Value		
		$TL = 1$	$TL = 10$	$TL = 30$
GA	P_{mut}	0.03	0.05	0.05
	$ Pop $	50	100	250
PSO	V_{max}	4	4	4
	c_1, c_2	2	2	2
	$ Pop $	50	100	250
COA	$MaxE_i$	2	2	2
	$MinE_i$	1	1	1
	$ Pop $	50	100	250

larger values for number of jobs, it would be much harder to solve the problems with the less number of machines.

5.3. Comparison of metaheuristic algorithms

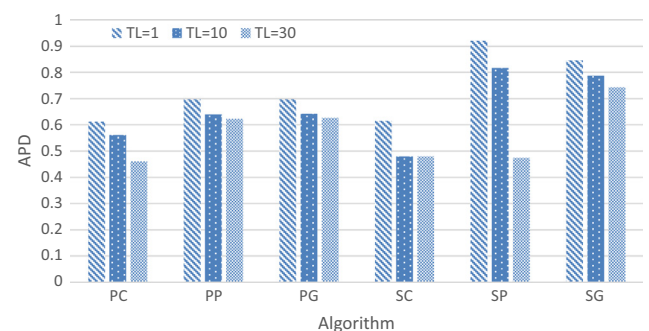
To analyze the performance of the developed algorithms, we have imposed CPU run time limits (TL) of 1, 10 and 30 s as the stopping criterion and obtain proper values for parameters of algorithms based on the design of experiments (DOE) techniques, as given in Table 10.

In what follows, the average percentage deviation (APD) of the obtained solutions by each algorithm (in a given time limit) from the best found solution available is demonstrated for various scales of problems.

As is evident in Fig. 12, in the small scale instances the performance of all the algorithms is improved by increasing the time limit and SC algorithms outperforms the other methods, abbreviated as given.

We can also compare the CPLEX performance with the developed metaheuristics based on the small size instances and with $TL = 30$ (For larger size instances and also for shorter run times, we do not have any feasible solution for many of the test instances). Computational results indicate that for 218 out of 240 small size test instances we have found at least a feasible solution and $APD = 0.04$ based on these test instances. This value of APD implies that CPLEX is much better in this special case rather than the metaheuristics but for harder cases, i.e. shorter run times and larger size, CPLEX is outperformed by the metaheuristics.

Considering the APD obtained for medium and large instances, see Figs. 13 and 14, the comparative analysis of algorithms shows the superiority of PC algorithms over the other approaches. It is to be noted that since the optimal solution is not achievable for the mentioned instances, APD is measured based on the best available solution, hence this value is not directly proportional to the size of

**Fig. 12.** Performance of algorithms for small size instances.

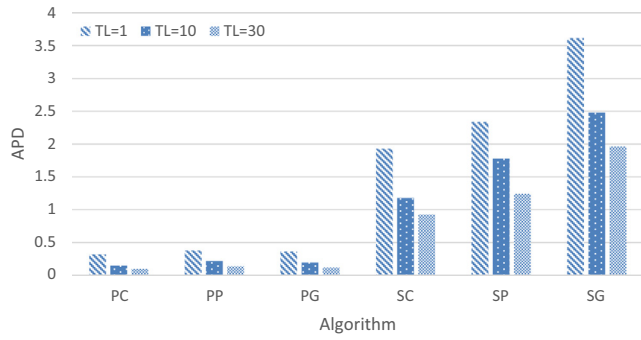


Fig. 13. Performance of algorithms for medium size instances.

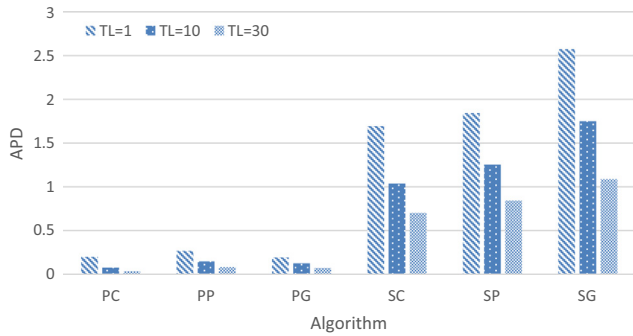


Fig. 14. Performance of algorithms for large size instances.

the problem. It is also concluded that larger TL leads to the less APD in all the three groups of instances.

Having studied the results, we can conclude that among the two schedule generation schemes, PSGS provides better performance than SSGS. Furthermore, the superiority of COA over other metaheuristic algorithms is inferred.

5.4. Statistical analyses

In this section, some statistical analysis is done in order to investigate the efficiency of the two schedule generation schemes. We also make a comparison between the metaheuristic approaches. Hence, for each instance and in each time limit, we have six solution for each instance, obtained by the algorithms SP, SC, SG, PP, PC and PG.

To evaluate the performance of SSGS and PSGS, we first compare the average of solutions reached by SG, SC and SP with the average of obtained solutions by PG, PC and PP. To do so, a paired- t test is used, in which the null hypothesis is defined for the equality of performance (makespan) of the two mentioned schemes, and the alternative hypothesis states the better performance (less makespan) of PSGS rather than SSGA. Having checked the data to be normally distributed, we use MINITAB to test the hypothesis for $TL = 1, 10, 30$. Since for the paired- t test P -value is reported as equal to P -value < 0.001 , the null hypothesis is rejected and the alternative hypothesis is accepted in its place.

To compare the efficiency of the metaheuristics, we have performed three paired- t tests in a similar manner illustrated in the previous paragraph. The concluded results, depicted in the Figs. 12–14, show the superiority of COA and the weaker performance of GA among this group of methods. For further explanation, let us first compare the efficiency of COA and PSO for $TL = 1$. Ignoring the results of GA, we are then given four solutions obtained by SP, SC, PP and PC. Thereafter, for each instance, we compare the mean

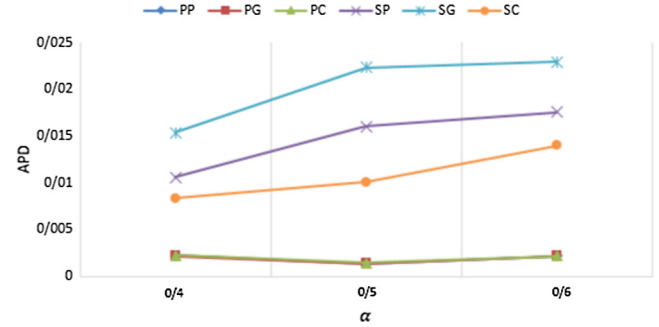


Fig. 15. Impact of α on APD.

obtained makespans of PP and PC with those obtained by applying SP and SC. Having considered the alternative hypothesis as the less makespan value of COA rather than PSO vs. the null one, denoting the equality of the two mentioned parameters, the aforementioned procedure is implemented in order to check the statistical test. By applying this test for $TL = 1, 10, 30$, we obtained P -value < 0.001 for each of the nine test. Consequently, we conclude that COA outperformed two other metaheuristics and also PSO has better performance than GA.

5.5. Impact of α parameter on APD

Computational results show that the value of α influences APD as well as the percentage of feasibility (POF), indicating the percentage generated feasible solutions by PSGS or SSGS. Figs. 15 and 16 depict the impact of this parameter on the mentioned measures.

Considering the trend observed in Fig. 15, it is concluded that the obtained results for the PSGS is not dependent on the parameter α and outperforms the SSGS for all values of α . On the other hand, SSGS shows better performance in the cases with smaller value of α than larger values of this parameter.

As presented in Fig. 16, except for the PP and SG, POF is 100% in the developed methods, while in the two mentioned ones, this measure decreases by increasing the value of α . The influence of the percentage of positive jobs on the complexity of the problem is also investigated in Van Belle et al. (2012), where a single machine scheduling problem subject to inventory constraints and regardless of the release dates is studied. Based on the comparative results, a higher percentage of positive jobs leads to harder instances to be solved.

5.6. Impact of $|M|$ on APD

To show the effect of the number of machines on the results, we conducted some computational experiments on the instances with

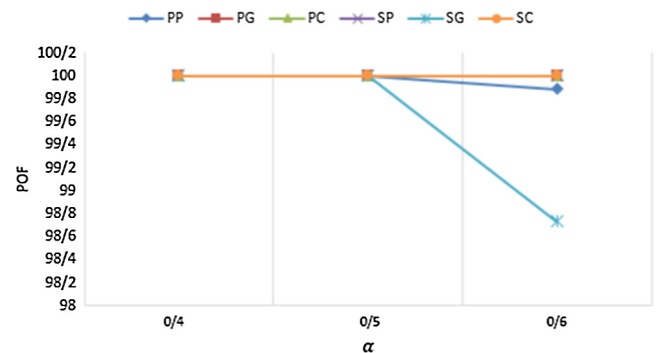
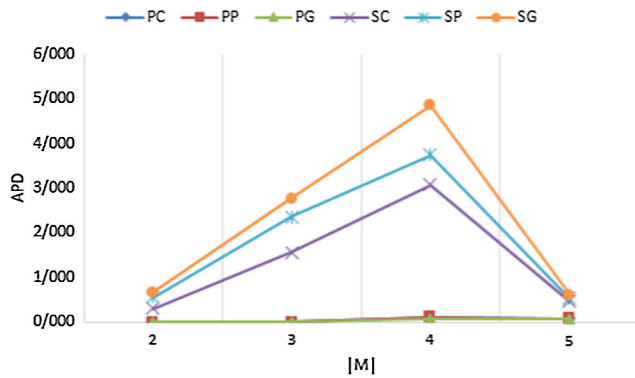
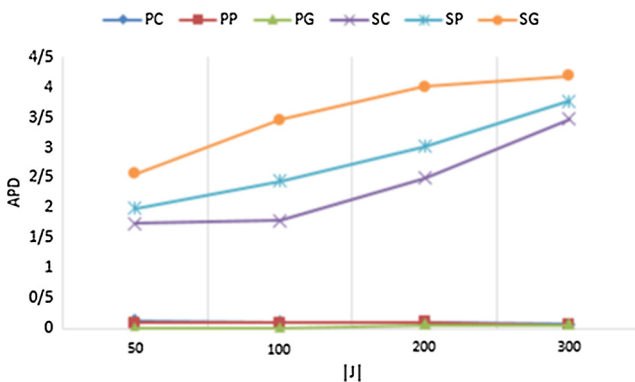


Fig. 16. Impact of α on POF.

Fig. 17. Impact of $|M|$ on APD.Fig. 18. Impact of $|J|$ on APD.

$|J| = 200$ and $\alpha = 50\%$. As provided in Fig. 17, the modifications of this parameter have no effect on APD obtained by applying PSGS, while in SSGS a considerable growth in APD is observed when $|M|$ is increased to 4.

5.7. Impact of $|J|$ on APD

Similar to the aforementioned analysis, we consider the instances with $|M| = 4$ and $\alpha = 50\%$ to analyze how far larger values for $|J|$ have an impact on APD. Fig. 18 shows that this parameter is an effective one in SSGS, so that the increasing amount of this parameter results in the larger values of APD.

6. Conclusions

In this paper, a transshipment scheduling problem with multiple identical stations has been discussed, where inventory and release date constraints are taken into account. This problem has been formulated as an integer linear programming model with an objective of minimization of makespan. Due to the complexity of the developed model, two constructive heuristics, named parallel and serial schedule generation schemes, and three metaheuristic algorithms have been developed based on GA, PSO and CO

approaches. Concluding our computational study shows that combination of the PSGS and the COA shows the best performance among all developed algorithms.

For future research, we believe the following directions to be promising. First, the exact methods can be applied for this problem to be solved to optimality. Second, generalizing the problem setting to more than one type of product can be motivated from real world applications in order to model a more realistic extension.

References

- Alpan, G., Bauchau, S., Larbi, R., & Penz, B. (2008). Optimal operations scheduling in a crossdock with multi strip and multi stack doors. *International Conference on Computers and Industrial Engineering* (Vol. 2, pp. 1168–1176).
- Alpan, G., Ladier, A. L., Larbi, R., & Penz, B. (2011a). Heuristic solutions for transshipment problems in a multiple door cross docking warehouse. *Computers & Industrial Engineering*, 61(2), 402–408.
- Alpan, G., Larbi, R., & Penz, B. (2011b). A bounded dynamic programming approach to schedule operations in a cross docking platform. *Computers & Industrial Engineering*, 60(3), 385–396.
- Boysen, N., & Fliehdner, M. (2010). Cross dock scheduling: Classification, literature review and research agenda. *Omega*, 38(6), 413–422.
- Boysen, N., Fliehdner, M., & Scholl, A. (2010). Scheduling inbound and outbound trucks at cross docking terminals. *OR Spectrum*, 32(1), 135–161.
- Briskorn, D., Choi, B. C., Lee, K., Leung, J., & Pinedo, M. (2010). Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2), 605–619.
- Briskorn, D., Jaehn, F., & Pesch, E. (2013). Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 16(1), 105–115.
- Briskorn, D., & Leung, J. Y. (2013). Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, 64(12), 1851–1864.
- Briskorn, D., & Pesch, E. (2013). Variable very large neighbourhood algorithms for truck sequencing at transshipment terminals. *International Journal of Production Research*, 51(23–24), 7140–7155.
- Debels, D., De Reyck, B., Leus, R., & Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism metaheuristic for project scheduling. *European Journal of Operational Research*, 169(2), 638–653.
- Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the sixth international symposium on micro machine and human science* (Vol. 1, pp. 39–43).
- Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, 287–326.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence*. Ann Arbor, MI: University of Michigan Press.
- Izadian, H., Ladani, B. T., Abraham, A., & Snasel, V. (2010). A discrete particle swarm optimization approach for grid job scheduling. *International Journal of Innovative Computing, Information and Control*, 6(9), 4219–4233.
- Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2), 320–333.
- MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1(14), pp. 281–297).
- Madani-Isfahani, M., Tavakkoli-Moghaddam, R., & Naderi, B. (2014). Multiple cross-docks scheduling using two metaheuristic algorithms. *Computers & Industrial Engineering*, 74, 129–138.
- Rajabioun, R. (2011). Cuckoo optimization algorithm. *Applied Soft Computing*, 11(8), 5508–5518.
- Ranjbar, M., De Reyck, B., & Kianfar, F. (2009). A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research*, 193(1), 35–48.
- Ranjbar, M. R., & Kianfar, F. (2007). Solving the discrete time/resource trade-off problem in project scheduling with genetic algorithms. *Applied Mathematics and Computation*, 191(2), 451–456.
- Vahdani, B., & Zandieh, M. (2010). Scheduling trucks in cross-docking systems: Robust metaheuristics. *Computers & Industrial Engineering*, 58(1), 12–24.
- Van Belle, J., Valckenaers, P., & Cattrysse, D. (2012). Cross-docking: State of the art. *Omega*, 40(6), 827–846.