JOURNAL OF INDUSTRIAL AND PRODUCTION ENGINEERING
https://doi.org/10.1080/21681015.2019.1647300

Taylor & Francis
Taylor & Francis Group

ARTICLE

Transshipment scheduling at a single station with release date and inventory constraints

Masoumeh Ghorbanzadeh, Mohammad Ranjbar and Negin Jamili

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

ABSTRACT

We study a loading and unloading scheduling problem in a single station transshipment terminal, in which inventory and release date constraints are subjected and the objective is to minimize the makespan. We propose an integer linear programming model; as well as a heuristic algorithm and two exact solution approaches: a dynamic programming algorithm and a brand-and-bound algorithm. Finally, performances of the developed algorithms are compared and analyzed using randomly generated test instances. Regarding the computational results, it is revealed that the simple developed heuristic algorithm is very fast and efficient. Also, the branch-and-bound algorithm outperforms the dynamic programming algorithm.

Abbreviations: Imperialist competitive algorithm (ICA); Variable neighborhood search (VNS); Genetic algorithm (GA): Adaptive tabu search (ATS); First-come–first served(FCFS); Branch-and-bound (B&B)

ARTICLE HISTORY
Received 17 January 2018
Accepted 16 July 2019

KEYWORDS
Scheduling; transshipment terminal; branch-and-bound algorithm

1. Introduction

Transshipment terminals are considered as intermediate nodes in a distribution network, where items are transferred without being stored for a long period of time. These centers are mainly known as cross-docking centers in supply chain systems. In general, cross docking is an important logistic strategy in which materials move directly from the receiving dock to the shipping one, without actually being held as inventory in a warehouse. Shipments typically stay less than 24 h, and even sometimes less than an hour, at the cross-dock [1].

According to what proposed by Yu and Egbelu [2], Figure 1 shows the flow of materials in a typical cross-docking operation that aims to eliminate storage and excessive material handling. So, a cross-dock is a consolidation point in a logistic transshipment network where multiple shipments from various suppliers are unloaded from inbound trucks, stored in a temporarily storage and then loaded onto outbound trucks based on their destinations to minimize the transportation costs.

There are many papers related to cross-docking in the literature. Boysen and Fliedner [3] and Buijs et al. [4] provided recent surveys of cross-docking literature. Generally, one of the objectives of cross-docking systems is to find an appropriate truck docking sequence for both inbound and outbound trucks to optimize some system performance measures such as costs and operation time. To simplify settings in practical situations, various assumptions are made in the existing literature. In some articles, only one dock has been considered for each receiving and shipping stage. For instance, Lee and Chen [5] studied a tow-machine cross-docking flow shop scheduling problem. They assumed that each inbound truck carries a specific type of product that may be needed by many destinations and the outbound trucks deliver several products with the same destinations. Arabani et al. [6] addressed a multi-objective cross-docking problem, in which items are unloaded in a single receiving dock and loaded onto one available shipping dock. In another study, the same authors developed their study by considering several receiving and shipping docks [7]. Yu and Egbelu [2] proposed a model where it is assumed that there is a temporary storage in front of the shipping dock to be able to store arriving products and some of these products may wait in the temporary storage until an appropriate outbound truck is available.

Addressing a problem with similar assumptions to the previously mentioned study, Forouharfard and Zandieh [8] and Ghobadian et al. [9] developed an imperialist competitive algorithm (ICA) and a GRASP method, respectively. Vahdani et al. [10] considered another cross-docking problem in which there is no temporary storage and trucks containing certain products are permitted to be held in order to come back to the corresponding dock and continue their task. In

CONTACT Mohammad Ranjbar m_ranjbar@um.ac.ir Department of Industrial Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

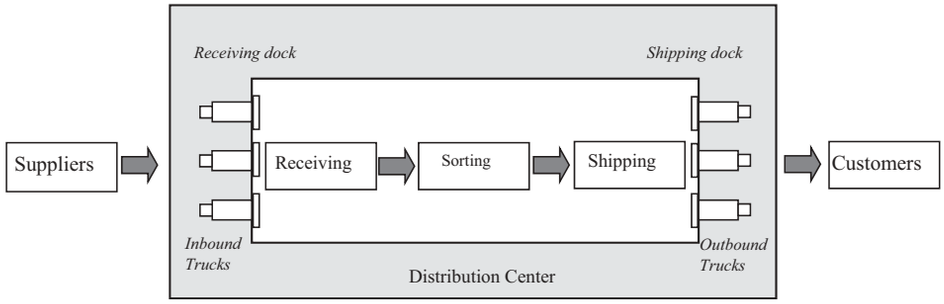

Figure 1. Typical flow in a cross-docking system.

[11] Vahdani and Zandieh applied five meta-heuristic algorithms to schedule trucks such that total operation times in the facility are minimized. Based on the obtained results, these authors recommend their variable neighborhood search (VNS) for scheduling trucks in cross-docking systems. Boloori Arabani et al. [12] and Alvarez-Perez et al. [13] developed a solution approach for cross-docking scheduling problem according to the just-in-time approach. Sadykov [14] focused on the operational activities at a cross-docking terminal with one receiving door, one shipping door, and a storage place. He proposed a mathematical model for optimizing the truck schedule for both inbound and outbound trucks while minimizing the storage cost. In the case of scheduling in cross-docking system in food industry, Boysen [15] worked on a problem where temporary storage is forbidden (zero inventory) and products have to be loaded into the shipping trucks immediately after being unloaded from the receiving ones.

Considering temporary storage, Boysen et al. [16] presented a base model for scheduling trucks as terminals. Konur and Golias [17] emphasized on scheduling of inbound trucks at the receiving doors of a facility, in case of uncertainty of arrival times. They formulate a bi-level optimization problem and a genetic algorithm (GA) to find a scheduling strategy which minimizes both handling and expected waiting times of the inbound trucks.

Moreover, there are studies focusing on cross-docking terminals with multiple dock doors. Finding an optimal assignment of trucks to the docks is crucially important as it strongly influences the performance of transshipment network. In this regard, some authors aim at deciding on the assignment of inbound docks, outbound docks or both at the same time. To the best of our knowledge, the model proposed by Miao et al. [18] is the first research where both inbound and outbound docks are taken into consideration at the same time. They presented a mathematical model and a few meta-heuristics to solve the assignment problem with the objective of minimizing both the total operational cost and the number of unfulfilled shipments. In another research, Miao et al. [19] extended the aforementioned problem and presented an adaptive tabu search (ATS). Moreover, we cite Chen and Song [20] who developed the model proposed in [5] by assuming at least one of the two stages of operations contains more than one parallel machine, and Van belle et al. [21] who simultaneously scheduled inbound and outbound trucks at several dock doors while minimizing two objectives: the total travel time and the total tardiness.

There are a group of studies in the literature of cross-docking scheduling considering different characteristics of inbound and outbound docks. Soltani and Sadjadi [22] considered a cross-docking system where no temporary storage is allowed. Making the assumption that both the receiving and the shipping trucks can move in and out of the docks during their tasks, these authors proposed two hybrid meta-heuristic to minimize the flow time of the system. In [23] Boysen et al. investigated an operational truck scheduling problem for cross-docking terminals with fixed outbound schedules. As the departure times are fixed, if a shipment is not loaded onto the designated outbound truck before its departure, the shipment's value is considered as lost profit. The objective is to schedule inbound trucks so that the total value of delayed shipments is minimized. Alpan et al. [24,25] and Maknoon [26] studied a cross-dock scheduling problem to determine a sequence for serving outbound trucks at multiple stack doors. They also consider a First-Come-First-Served (FCFS) policy to assign the order of incoming trucks. In a recently published article, Mohtashami [27] defined two possible scenarios for a dock holding pattern and outbound trucks while considering a temporary storage. He presents a genetic algorithm-based approach to find an appropriate sequence of trucks that minimizes the make-span. In the research of Shakeri et al. [28] and Hermel et al. [29], a resource-constrained cross-dock was described where the problem is to decide the scheduling of the receiving and shipping trucks subject to limited resources.

Another group of studies in this area focus on the truck scheduling problem at a terminal with a single

door for both incoming and outgoing trucks. The simplified version of this problem consists of homogeneous products. Briskorn et al. [30] dealt with the problem of scheduling trucks at a transshipment terminal with an only one gate and determine the computational complexity of the problem and its counterparts with different objectives. In [31] Briskorn et al. extended the problem introduced in [30] and design a branch-and-bound and a dynamic programming algorithm to obtain a solution with the minimum total weighted completion time. Briskorn and Leung [32] also considered the mentioned problem and provide a number of heuristic methods that find valid upper bounds for any branch-and-bound algorithm that is based on branching in both forward and backward directions, in order to find a schedule of jobs such that maximum lateness is minimized. Employing a local search strategy, Briskorn and Pesch [33] worked on transshipment terminals where inventory constraints must be fulfilled so that the total pick up delivery and the total delivery quantity are non-negative and do not exceed the inventory's capacity. Also, Bazgosha et al. [34] developed two schedule generation schemes and three metaheuristic algorithms for scheduling of loading and unloading operations in a multi-station transshipment terminal with release dates and inventory constraints.

In this paper, we develop the model presented in [33] and take release dates for inbound trucks into consideration. These release dates can be seen as a reasonable approximation of reality, i.e., receiving trucks are not available for processing at the beginning of the planning horizon and their arrival times are assumed to be different. Following well-known three field notation introduced in [35], this problem can be represented as $1|r_j,inv|C_{max}$. It is to be noted that the algorithms applied in the published literature cannot be easily modified to be used in this study and thus we propose new approaches to solve instances of the proposed problem. Although the solution approaches developed by Bzgosha et al. [34] can be applied to our problem, we are seeking optimal solution approaches while they do not have such property.

The contribution of this paper is threefold: (1) we develop a simple yet very fast and efficient heuristic algorithm for $1|r_j,inv|C_{max}$; (2) we develop a dynamic programming algorithm that solves the instances of the problem until optimality; and (3) we develop a branch-and-bound algorithm that solves $1|r_j,inv|C_{max}$.

The remainder of this paper is organized as follows. In Section 2, we formulate the problem as a mixed linear integer program and in Section 3 we develop a set of solution approaches. The proposed algorithms are evaluated via computational experiments in Section 4. Finally, conclusions are drawn and future research directions are discussed in Section 5.

2. Problem description and modeling

This section addresses how the problem is modeled by introducing the main assumptions of the problem as well as the notations for proposed mathematical model. We assume that there is a single dock at a transshipment terminal, in which both loading and unloading operations are carried out, such that only one truck has to be served at each moment. This problem corresponds to a classical single machine scheduling where the machine corresponds the transshipment terminal and each job corresponds to either a loading or an unloading operation. Also, we consider only one product type in this research. Thus, a loading (an unloading) job includes loading (unloading) a known number of the product to (from) a truck. Also, each truck carries only one job at a time. The following assumptions are applicable to the model:

(a) Number of trucks are infinite and they are available only after their arrival times.
(b) Preemptions are not allowed. In other words, loading and unloading activities have to be done with no interruption.
(c) An initial inventory level is defined for the terminal. Inbound and outbound shipments quantity must be taken into account such that in all times, the inventory level does not exceed its capacity and remains non-negative.
(d) After the completion of each loading (unloading) job, the inventory level will decrease (increase).

The required parameters and decision variables used for describing the model is given in Table 1.

We define the binary decision variable X_{jt}which is 1 if processing of job j is completed at time t and 0, otherwise. It should be noted that truck j is an inbound or outbound truck if $\delta_j > 0$ or $\delta_j < 0$, respectively.

For example, consider an instance with $n = 5$, where $I_{in} = 20$ and $I_c = 21$. Table 2 represents the

Table 1. Description of parameters.

Notations	Definitions
$J = \{1, 2, \ldots, n\}$	Set of loading and unloading jobs with indices i and j.
$T = \{s^*, \ldots, \lvert T\rvert\}$	The time slots available for (un-)loading trucks/ jobs with index t, where $s^* = \min_{j\in J}\{r_j + p_j\}$.
I_{in}	Initial inventory level specified by the number of product units.
I_c	Capacity of Inventory specified by the number of product units.
I_t	Inventory of transshipment terminal at time instant t.
p_j	Processing time of job j.
r_j	Release date of job j.
δ_j	Inventory modification made by processing job j.
$\lvert T\rvert$	An upper bound for completion time of all jobs.

Table 2. Information of the instance.

j	1	2	3	4	5
p_j	5	3	6	3	7
r_j	0	8	15	1	8
δ_j	−10	+1	+7	−9	−7

information required for this problem. In Figure 2 a feasible scheduling for this example is depicted as a Gantt chart with $C_{max} = 28$

The mathematical formulation of $1|r_j, inv|C_{max}$ reads as follows.

$$MinC_{max} \tag{1}$$

Subject to:

$$\sum_{t=s^*}^{|T|} X_{jt} = 1 \qquad \forall j \in J \tag{2}$$

$$\sum_{\tau=t-p_j+1}^{t} \sum_{i \in J \setminus \{j\}} X_{i\tau} \leq M\left(1 - X_{jt}\right) \quad \forall j \in J, \forall t \in T \tag{3}$$

$$\sum_{t=s^*}^{|T|} tX_{jt} - p_j \geq r_j \quad \forall j \in J \tag{4}$$

$$C_{max} \geq \sum_{t=s^*}^{|T|} tX_{jt} \quad \forall j \in J \tag{5}$$

$$I_{s^*} = I_{in} \tag{6}$$

$$I_t = I_{t-1} + \sum_{j \in J} \delta_j X_{jt} \qquad \forall t \in T \tag{7}$$

$$I_t \leq I_c \qquad \forall t \in T \tag{8}$$

$$X_{jt} \in \{0, 1\} \quad \forall j \in J, \forall t \in T \tag{9}$$

$$I_t \in \mathbb{Z}^+ \quad \forall t \in T \tag{10}$$

The objective function (1) minimizes the makespan. Constraints (2) ensures that each job is assigned to be executed only once and constraints (3), in which M is a sufficiently large positive value, guarantees that at each time moment only one job can be processed. Constraints (4) indicates that the start time of each job must be greater than or equal to its release date. Restriction imposed to makespan is defined in Constraints (5). Constraint (6) and constraints (7) are related to the inventory level and stipulate the amount of inventory at each moment. The storage quantities are bounded in Constraints (8) and cannot be more than I_c. Finally, the last two sets of constraints describe the type of the variables in which $\mathbb{Z}^+$ represents the set of non-negative integers.

Solving the mathematical model for the previously presented instance, the optimal schedule, which is depicted in Figure 3, has an objective value of 27.

3. Solution approaches

Although the formulation of this problem is presented as an integer linear programming model, it is only

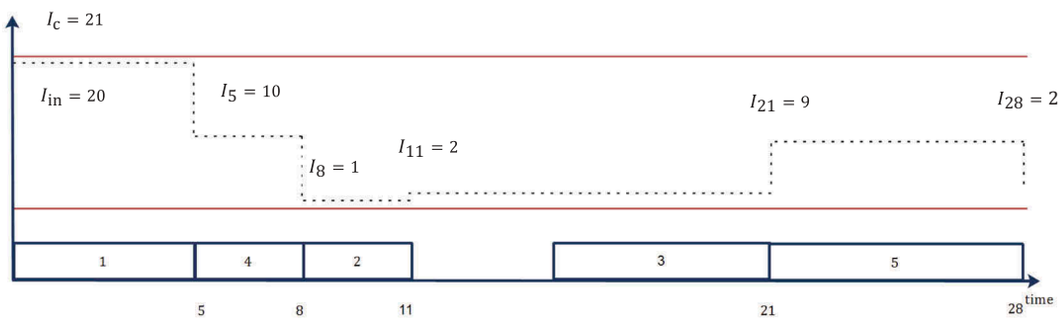

Figure 2. A feasible scheduling for the example problem.

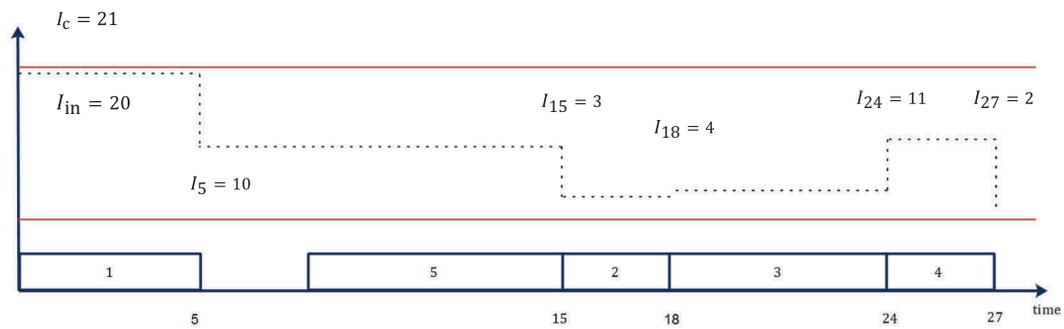

Figure 3. The optimal scheduling for the example problem.

applicable for small instances and its complexity makes it very time-consuming and impractical for real size problems. Since this problem is NP-hard [36], we develop a heuristic and two exact methods to solve the problem.

In the following, a heuristic algorithm and two exact solution methods based on dynamic programming and branch-and-bound algorithms are provided to solve the problem.

3.1. Heuristic algorithm

The designed heuristic algorithm that is described in this section is able to obtain an appropriate feasible schedule for the proposed cross-docking problem in terms of both quality and runtime, even for large-scale instances. Despite its short required runtime, sometimes this method is unable to reach to optimal or near-optimal solutions. Also, in a few cases, it is not able to find even a feasible solution. Table 3 summarizes the notations applied in this algorithm.

The steps of this algorithm for $G = C_{max}$ are demonstrated in Algorithm 1. In Step 2, eligible jobs are defined as those which do not violate the inventory constraints. Step 3 is implemented to choose a job in a way that the total idle time of machine before and after that job is minimized. In order to calculate the gap after each job j (gap_j^A), we examine all possible jobs which may be inserted in the schedule in the next iteration. Also, it should be mentioned that operator $Arg(\ \dots\ )$ defines an index where the value of $min_j(gap^j)$ is minimized.

INPUT: an instance of $1|r_j, inv|C_{max}$

Step 1. Initialize data: Set $t = 0$, $R_t = \{1, \dots, n\}$, $E_t = \emptyset$, $S_t = \emptyset$, $I_t = I_{in}$ and $G = 0$;

Step 2. Set $E_t = \{i | i \in R_t, 0 \le I_t + \delta_i \le I_C\}$
if $E_t = \emptyset$ and $R_t \neq \emptyset$ **then**
Stop; {no feasible solution can be found}
else if $|R_t| = |E_t| = 1$ **then**
select $j^* \in E_t$ and go to Step 4;
else go to Step 3;

Step 3. **for all** $j \in E_t$ **do**
let $\tau_j = \max(t, r_j)$, $I_t^j = I_t + \delta_j$, $E_t^j = \{i | i \in R_t \backslash \{j\}, 0 \le I_t^j + \delta_i \le I_C\}$ and $gap_j^B = \tau_j - t$;
end for
if for all $j \in E_t$: $E_t^j = \emptyset$ **then**
Stop; {no feasible solution can be found}
Else for all $i \in E_t^j$ **do**

$$gap_j^A = \min_{\forall i \in E_t^j} \begin{pmatrix} \max(\tau_j + p_j, r_i) - \\ (\tau_j + p_j) \end{pmatrix};$$

$g^j = gap_j^B + gap_j^A$
end for
select $j^* = Arg\left(\min_{\forall j}(gap^j)\right)$;

Step 4. Let $\tau = \max(t, r_{j^*}) + p_{j^*}$, $R_\tau = R_t \backslash \{j^*\}$, $S_\tau = S_t \cup \{j^*\}$, $t = G = \tau$ and $I_t = I_t^j$;
if $R_t \neq \emptyset$ **then**
go to Step 2;
else return G and Stop.

Table 3. Notations and variables used in the heuristic algorithm.

Notations	Definitions
S_t	Set of scheduled jobs by the time of t.
R_t	Set of unscheduled jobs at time t.
E_t	Set of eligible jobs to start processing at time t.
gap_j^B	the interval time between the start time of job j and the completion time of its immediate previous job.
gap_j^A	the interval time between the completion time of job j and the start time of its immediate next job.
G	Objective value.

Algorithm 1: Pseudo-code of the proposed heuristic algorithm.

If we apply this algorithm to the example described in Section 2, we obtain the sequence depicted in Figure 2 with the objective value of 28. In Table 4, the first two iterations of this algorithm have been illustrated.

3.2. Dynamic programming algorithm

Dynamic programming is considered as an exact algorithm for solving complex problems by employing a combination of sequential decisions and improve the efficiency of calculations. According to the notations given in Table 5, we now present a straightforward dynamic programming algorithm for the proposed problem, where $G = C_{max}$.

We introduce dummy job 0 with $P_0 = 0$, $r_0 = 0$ and $\delta_0 = 0$, which is always scheduled as the first task. Let I_{in} be the inventory level of the terminal at time 0, the initial conditions of this algorithm are: $S = \emptyset, G(S) = 0$. Now, we can formulate the recursive function as:

$G(S) = \min_{j \in E(S \backslash \{j\})}(\max(G(S \backslash \{j\}), r_j) + p_j)$ And the stopping criterion is defined as $S = J$.

For example, consider the instance proposed in Section 2. Table 6 indicates all implementation steps of applying our developed dynamic programming to this instance in which symbol "✗" signifies infeasible solutions. Also, we assume $\Delta = \delta_j + \sum_{k \in S \backslash \{j\}} \delta_k + I_{in}$. This instance has several optimal solutions with objective function value 27 where our developed dynamic programming obtained the optimal sequence 1–2-5–3–4.

In the first step, we consider $S = \{1\}$ that signifies job 1 is the last completed job. This partial solution, resulting in $G(S) = 5$, is feasible because it corresponds the feasible value of $\Delta = 10$. Similarly, if set S includes one of the jobs 2, 4 and 5, we obtain the partial feasible solutions with objective function values $G(S) = 11, 4 \text{and} 15$, respectively. If we consider $S = \{3\}$, the value of Δ is 27 which is infeasible. In the second step, there are 10 different states for set S. For instance, if we assume $S = \{1, 4\}$, we get a feasible partial solution with $\Delta = 1$. In this case, the last completed job is either job 1 or job 4.

Table 4. Implementation of the heuristic algorithm.

Iteration 1:
Step 1. $t = 0, R_t = \{1,2,3,4,5\}, E_0 = \emptyset, S_0 = \emptyset, I_0 = 20, G = 0.$
Step2. $E_0 = \{1,2,4,5\}$
Step 3. $\tau_1 = 0, \tau_2 = 8, \tau_4 = 1, \tau_5 = 8.$
$I_0{}^1 = 10, I_0{}^2 = 21, I_0{}^4 = 11, I_0{}^5 = 13.$
$E_0{}^1 = \{2,3,4,5\}, E_0{}^2 = \{1,4,5\}, E_0{}^4 = \{1,2,3,5\}, E_0{}^5 = \{1,2,3,4\}.$
$gap_1^B = 0, gap_2^B = 8, gap_4^B = 1, gap_5^B = 8.$
$gap_1^A = 0, gap_2^A = 0, gap_4^A = 0, gap_5^A = 0.$
$gap^1 = 0, gap^2 = 8, gap^4 = 1, gap^5 = 8.$
$j^* = 1.$
Step 4. $\tau = 5, R_5 = \{2,3,4,5\}, S_5 = \{1\}, G = t = 5, I_5 = 10.$

Iteration 2:
Step2. $E_5 = \{2,3,4,5\}$
Step 3. $\tau_2 = 8, \tau_3 = 15, \tau_4 = 5, \tau_5 = 8.$
$I_5{}^2 = 11, I_5{}^3 = 17, I_5{}^4 = 1, I_5{}^5 = 3.$
$E_5{}^2 = \{3,4,5\}, E_5{}^3 = \{2,4,5\}, E_5{}^4 = \{2,3\}, E_5{}^5 = \{2,3\}.$
$gap_2^B = 3, gap_3^B = 10, gap_4^B = 0, gap_5^B = 3.$
$gap_2^A = 0, gap_3^A = 0, gap_4^A = 0, gap_5^A = 0.$
$gap^2 = 3, gap^3 = 10, gap^4 = 0, gap^5 = 3.$
$j^* = 4.$
Step 4. $\tau = 8, R_8 = \{2,3,5\}, S_5 = \{1,4\}, G = t = 8, I_8 = 1$

Table 5. Notations and variables used in the dynamic programming.

Notations	Definitions
S	Set of scheduled jobs.
$E(S)$	Set of eligible jobs to be added to S $(E(s) = \{j \mid j \notin S, 0 \le \delta_j + \sum_{k \in S} \delta_k + I_{in} \le I_c\})$.
$G(S)$	The optimal value of objective G for S.

If job 1 is completed after job 4 $(j = 1)$, the recursive function value taken from the previous step is $G(S\{4\}) = 4$ and it is updated as $G(S\{1,4\}) = \max(4,0) + 5 = 9$. Also, if, job 4 is the last completed job $(j = 4)$, the recursive function value taken from the previous step is $G(S\{1\}) = 5$ and is updated as $G(S\{1,4\}) = 8$. This indicates, if the two last unscheduled jobs are jobs 1 and 4, we have to schedule job 1 before job 4. In the third and fourth steps, there are 10 and 5 different states for set S, respectively. Eventually, in Step 5, we find two optimal solutions with optimal value of 27. In the first optimal solution, job 4 is the last completed job. Consequently, we have to back to Step 4 with sate S ={1,2,3,5}. This specifies the fourth completed job have to be either job 2 or job 3. For each case, we have to back to the corresponding previous state and find the optimal solution. Finally, we find three optimal solutions 1–5-3–2-4, 1–-5-2–3-4 and 1–5-3-4-2.

3.3. Branch-and-bound algorithm

Branch-and-bound (B&B) is an exact algorithm for discrete and combinatorial optimization problems. This method is designed based on exploring the solution space by enumeration of candidate solutions. The proposed algorithm in this paper provides an optimal scheduling of a given set of jobs by applying a branching strategy to create nodes and investigating all the solution candidates, consequently. A diagram showing this process is called a search tree, where each node indicated a sub-problem with a corresponding partial sequence of jobs. To avoid spending time on searching for infeasible or relatively poor solutions, we restrict the branching scheme using bounding procedure.

3.3.1. The branching scheme

To develop the search tree, we consider depth-first strategy in which exploring begins from the left-hand side of the search tree. Completing a branch to the end, we then backtrack and continue searching other nodes.

In this problem, choosing a node specifies a scheduled job. It is necessary for all the child nodes to be feasible, i.e., if the partial sequence implied by a node violates the inventory constraints, the related node is fathomed and other branches are regarded to continue exploring. It is to be mentioned that child nodes are sorted based on their r_j and branching starts from the left one which has the minimum value of r_j. The same strategy if followed for the nodes in the remaining levels.

Figure 4 depicts the search tree for the example problem. Each node is represented by two numbers; the first one denotes the node number and the second stands for the job scheduled in the sequence. In this figure, the optimal sequence path is highlighted by the red color.

Generally, search tree starts with a sequence which is currently scheduled. In this tree, each level is related to a job which has to be added to the sequence and the completed schedules are determined at the final level. Regarding the sequence defined by node 1–1 and 2–2, the objective value for this partial solution is C_{max}^{2-2} =11, where C_{max}^{l} determines the length of partial sequence associated with lth node. Since the currently available inventory at terminal is 11 products, the feasibility is guaranteed and other nodes are

Table 6. Solving the example problem using dynamic programming.

Step 1	S	1	2	3	4	5
	Δ	10	21	27	11	13
	j	1	2	x	4	5
	$G(S\backslash\{j\})$	0	0		0	0
	$G(S)$	5	11		4	15

Step 2	S	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}
	Δ	11	17	1	3	28
	j	1 2	1 3	1 4	1 5	X
	$G(S\backslash\{j\})$	11 5	X 5	4 5	15 5	
	$G(S)$	16 11	21	9 8	20 15	
	S	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
	Δ	12	14	18	20	4
	j	2 4	2 5	3 4	3 5	4 5
	$G(S\backslash\{j\})$	4 11	15 11	4 X	15 X	15 4
	$G(S)$	11 14	18 18	21	21	18 15

Step 3	S	{1,2,3}	{1,2,4}	{1,2,5}	{1,3,4}	{1,3,5}
	Δ	18	2	4	8	10
	j	1 2 3	1 2 4	1 2 5	1 3 4	1 3 5
	$G(S\backslash\{j\})$	X 21 11	11 8 11	18 15 11	21 8 21	21 15 21
	$G(S)$	24 21	16 11 14	23 18 18	26 21 24	26 21 28
	S	{1,4,5}	{2,3,4}	{2,3,5}	{2,4,5}	{3,4,5}
	Δ	−6	19	21	5	11
	j	X	2 3 4	2 3 5	2 4 5	3 4 5
	$G(S\backslash\{j\})$		21 11 X	21 18 X	15 18 11	15 21 21
	$G(S)$		24 21	24 24	18 21 18	21 24 28

Step 4	S	{1,2,3,4}	{1,2,3,5}	{1,2,4,5}	{1,3,4,5}	{2,3,4,5}
	Δ	9	11	−5	1	12
	j	1 2 3 4	1 2 3 5	X	1 3 4 5	2 3 4 5
	$G(S\backslash\{j\})$	21 21 11 21	24 21 18 21		21 X 21 21	21 18 24 21
	$G(S)$	26 24 21 24	29 24 24 28		26 24 28	24 24 27 28

Step 5	S	{1,2,3,4,5}
	Δ	0
	j	1 2 3 4 5
	$G(S\backslash\{j\})$	24 24 X 24 21
	$G(S)$	29 27 27 28

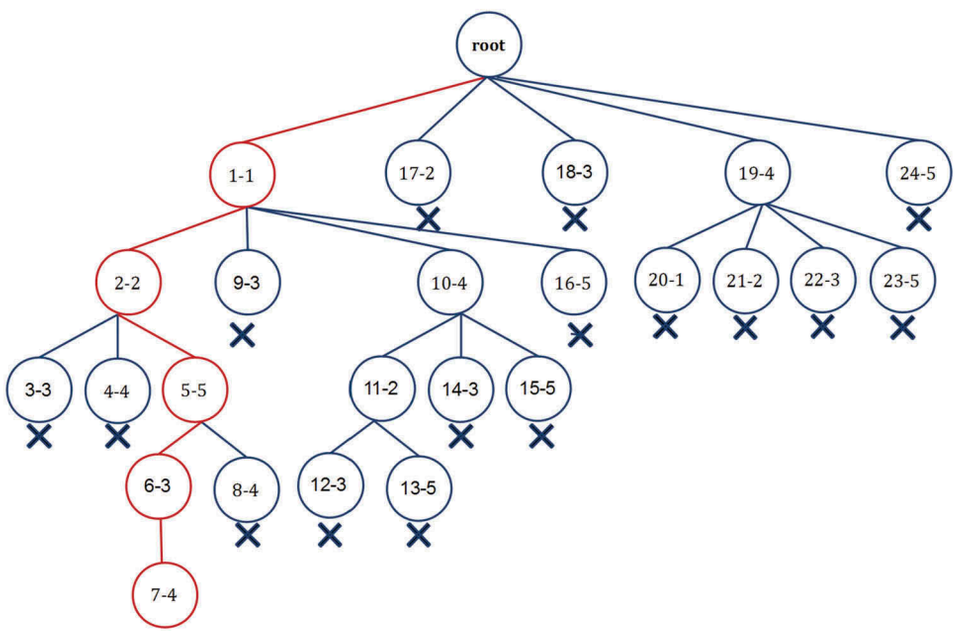

Figure 4. Search tree of the example problem.

created so that a longer sequence with a bigger objective function is provided. Consider another sequence consisting of nodes 1–1, 10–4 and 15–5. For this partial solution $C_{max}^{15-5} = 15$ and the inventory level is −6. Thus, the node 15–5 is fathomed based on the negative amount of inventory.

Since the developed heuristic algorithm results in a sequence with $C_{max} = 28$, we start our search using B&B algorithm with $UB = 28$. The most left path of the search tree including nodes ⟨1–1, 2–2, 3–3⟩ is fathomed due to ordinary rule $LB_{3-3} \geq UB$ where LB_l for each node l will be described in subsequent sections. Also, the paths including nodes ⟨1–1, 2–2, 4–4⟩, ⟨19–4, 20–1⟩ are cut because of *DR*1, described in Section 3.4. Next, the path containing nodes ⟨1–1, 2–2, 5–5, 6–3, 7–4⟩ concludes a feasible solution with $C_{max} = 27$ and consequently the upper bound is updated to 27. In continue, paths including nodes ⟨1–1, 2–2, 5–5, 8–4⟩, ⟨1–1, 10–4, 11–2, 13–5⟩, ⟨1–1, 10–4, 15–5⟩ are bounded due to negative inventory level. Moreover, the path including node 18–3 is cut because the inventory level exceeds the inventory capacity. Other nodes are fathomed due to the rule $LB_l \geq UB$.

3.3.2. Bounding scheme

In this algorithm, it is critically important to reduce the size of the search tree. In this regard, only the potential nodes for reaching to the optimal solution are branched.

3.3.2.1. Upper bound.

An upper bound is aimed to find a feasible solution in a very short time period. This solution is not essentially the optimal one and will be updated at any time a better solution is reached. In this approach, C_{max} of the solution of heuristic algorithm described in Section 3–1 is considered as the initial upper bound, shown by *UB*. Whenever a complete sequence results in a smaller value of C_{max} rather than the upper bound, *UB* will be updated by the achieved objective value.

We should mention that in the cases when the heuristic algorithm fails to obtain a solution, we consider $UB = \infty$.

3.3.2.2. Lower bound.

In this algorithm, a lower bound is designed to get a solution which is scheduled in ideal condition. Clearly, this solution may not be feasible since some restrictions such as inventory constraints are not imposed. Considering node l, the lower bound for this node is represented by LB_l and computed as $LB_l = max\left(LB_l^1, LB_l^2\right)$. We consider $LB_l^1 = \max\left(C_{max}^l, \min_{j \in R_l}\left(r_j\right)\right) + \sum_{j \in R_l} p_j$ while LB_l^2 is obtained by imposing the partial obtained solution to the relaxation of the linear integer model described in Section 2.

3.4. Dominance rules

Dominance rules are additional features of the B&B algorithm exploited to reduce the computational burden by omitting some nodes which are not possible to cause good quality results. As a basic commonplace rule in all B&B algorithms, in each node l, if we have $LB_l \geq UB$, then this node is fathomed. Also, we develop the following dominance rules (*DR*).

*DR*1: In each node l, if exchanging the last two consecutive jobs leads to a better feasible solution, then this node is fathomed.

For instance, in Figure 5, swapping the last two jobs in the sequence assigned by nodes 1–1, 2–2 and 4–4 leads to a feasible schedule without violating

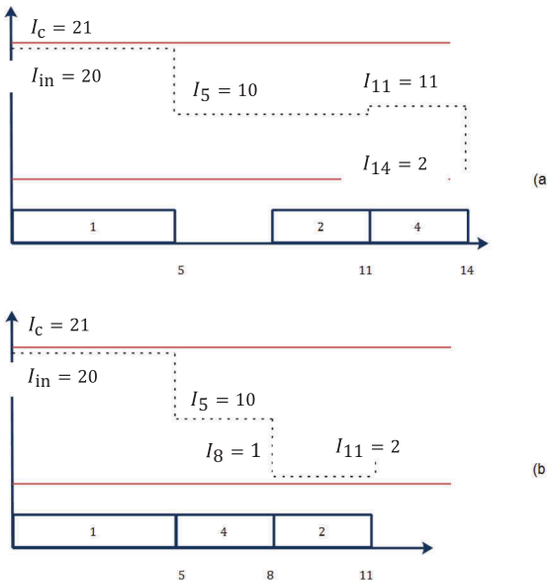

Figure 5. An example of DR1.

an inventory constraint. As the partial objective value of the modified schedule is decreased from 14 to 11, branching is not executed for node 4–4.

4. Computational results

To evaluate the performance of the developed algorithms, we have generated 540 test problems in two groups of small and large size instances. Thereafter, the algorithms are implemented in Visual C++ and numerical results are obtained using a computer with an Intel Core i5 and 4GB of RAM. In addition, we employed IBM CPLEX 12.6.1 as a solver for the integer linear programming model.

4.1. Experimental setup

We created small and large instances with (20, 25, 30) and (50, 75, 100) jobs, respectively. Due to the conclusion obtained by different experiments, runtime of the instances is significantly affected by varying p_j and r_j. Therefore, the processing time and release date of each job (p_j) is chosen randomly from integer values in $[1,10]$, $[1,20]$ and $[1,30]$. Moreover, the integer release dates of jobs (r_j) are uniformly distributed in the intervals $\left[0, 0.05 \times \sum_j p_j\right]$, $\left[0,\ 0.75 \times \sum_j p_j\right]$ or $\left[0,\ \sum_j p_j\right]$. Generating 10 random instances for each combination of parameters n, p_j and r_j, we have totally 540 test instances to conduct a comparison between the algorithms. Other parameters follow a specific uniform distribution as shown in Table 7.

It should be noted that α denotes the percentage of positive jobs in each instances. Also, the distributions of l_{in} and l_C have been chosen such that all generated solutions have feasible solutions.

4.2. Analysis of the results

In this section, the results of B&B algorithm and dynamic programming are presented in terms of CPU runtimes. For a fair comparison of all the developed methods, the identical time limit $TL = 1800$ seconds is imposed. In the following, the average CPU run times of the three approaches for the small instances are given based upon the second.

Table 7. Uniform distribution of parameters.

Parameter	Distribution
l_{in}	$\sim U\left[\max\left(0, -\sum_{j\in J^-}\delta_j\right), -\sum_{j\in J^-}\delta_j\right]$
l_C	$\sim U\left[l_{in} + \max\left(0, \sum_{j\in J}\delta_j\right), l_{in} + \sum_{j\in J^+}\delta_j\right]$
α	50%
δ_j	$\sim U[1,10]$

Table 8. The average CPU runtime of the mathematical model (seconds).

r_j \ n	p_j	20	25	30
$\left[0, 0.5 \times \sum_j p_j\right]$	$[1,10]$	9.164	41.79	103.538
	$[1,20]$	245.183	939.961	959.395
	$[1,30]$	1125.909	1432.360	1800
$\left[0, 0.75 \times \sum_j p_j\right]$	$[1,10]$	5.669	28.197	57.782
	$[1,20]$	77.697	565.310	1188.128
	$[1,30]$	561.4291	1277.375	1800
$\left[0,\ \sum_j p_j\right]$	$[1,10]$	5.890	12.085	207.325
	$[1,20]$	434.444	675.666	658.693
	$[1,30]$	433.729	736.286	1017.023

In order to solve the model presented in Section 2 using CPLEX for each instance, we consider $|T|$ as the objective value obtained from the developed heuristic algorithm. As provided in Table 8, increasing the number of jobs (n) and processing times (p_j) leads to larger CPU run time of the mathematical model, while r_j is not an effective parameter on this measure. It should be mentioned that the results of Table 8 are related to only those instances that could be solved optimally within the given time limit.

Table 9 shows the number of total instances solved within the time limit. According to results shown in this table, the number of unsolved instances grows by increasing the two parameters of n and p_j.

The results of dynamic programming algorithm are presented in Table 10. We see that all the small-size instances are solved by this method within the given time limit. It is also concluded that n is directly linked

Table 9. The number of total instances solved optimally by the mathematical model.

r_j \ n	p_j	20	25	30
$\left[0, 0.5 \times \sum_j p_j\right]$	$[1,10]$	10	10	10
	$[1,20]$	10	10	3
	$[1,30]$	10	4	0
$\left[0, 0.75 \times \sum_j p_j\right]$	$[1,10]$	10	10	10
	$[1,20]$	10	8	6
	$[1,30]$	10	6	0
$\left[0,\ \sum_j p_j\right]$	$[1,10]$	10	10	10
	$[1,20]$	10	10	6
	$[1,30]$	10	9	3

Table 10. The average CPU runtime of the dynamic programming algorithm (seconds).

r_j \ n	p_j	20	25	30
$\left[0, 0.5 \times \sum_j p_j\right]$	$[1,10]$	0.355	15.147	627.437
	$[1,20]$	0.397	10.981	572.898
	$[1,30]$	0.230	14.129	632.565
$\left[0, 0.75 \times \sum_j p_j\right]$	$[1,10]$	0.360	18.248	679.469
	$[1,20]$	0.422	10.738	538.697
	$[1,30]$	0. 386	9.714	636.748
$\left[0,\ \sum_j p\right]$	$[1,10]$	0.467	13.065	590.347
	$[1,20]$	0.279	10.192	488.454
	$[1,30]$	0.352	10.740	618.898

Table 11. The average CPU runtime of the B&B algorithm (seconds).

r_j	p_j \ n	20	25	30
$\left[0, 0.5 \times \sum_j p_j\right]$	$[1, 10]$	0.355	15.147	627.437
	$[1, 20]$	0.397	10.981	572.898
	$[1, 30]$	0.230	14.129	632.565
$\left[0, 0.75 \times \sum_j p_j\right]$	$[1, 10]$	0.360	18.248	679.469
	$[1, 20]$	0.422	10.738	538.697
	$[1, 30]$	0. 386	9.714	636.748
$\left[0, \sum_j p_j\right]$	$[1, 10]$	0.467	13.065	590.347
	$[1, 20]$	0.279	10.192	488.454
	$[1, 30]$	0.352	10.740	618.898

Table 12. The results of the heuristic algorithm.

	20	25	30
#Opt	86	83	83
APD	2.36	3.36	2.04

to the CPU runtime, whilst changing the value of r_j and p_j results in no further modification in the solving CPU runtime. It is worth noting that since this algorithm's CPU runtime increases dramatically with the increasing of problem size, it is not applicable for the large-size problems.

Regarding the results of B&B algorithm, shown in Table 11, parameter r_j can be considered as an effective parameter in this method, so that the larger value of this parameter leads to the longer CPU runtimes. In the case of large values of p_j, increasing r_j contributes to a growth in the CPU runtime.

Table 12 summarizes the results of the heuristic approach. For each group of instances, *#Opt* indicates the total number of instances that have been solved optimally by the heuristic algorithm. For those instances that the heuristic algorithm could not reach the optimal solutions, the values of $APD = \frac{\text{obtained solution} - \text{optimal solution}}{\text{optimal solution}} \times 100$, indicating the average percent of deviation from the optimal solution, have been shown. The results illustrate the efficiency of this algorithm.

Additionally, large-size instances are solved by the B&B algorithm (with *TL*=1800) as well as heuristic algorithm. Since most of the instances in this category are not solved optimally, we define *best solutions* instead of *optimal solutions* here. In Table 13, *#Best* denotes the number of instances where the heuristic algorithms could find the best solutions. For those instances that the heuristic algorithms could not find the best solutions, average percent deviation from best solutions is presented by *APD*.

Table 13. Results of the heuristic algorithm in large-size instances.

	50	75	100
APD	3.06	2.87	0.7
#Best	86	85	87

Table 14. Impact of the heuristic algorithm in the B&B algorithm.

r_j	p_j \ n	20	25	30
$\left[0, 0.5 \times \sum_j p_j\right]$	$[1, 10]$	11,608%	299,100%	179,560%
	$[1, 20]$	25,008%	34,500%	290,766%
	$[1, 30]$	29,460%	60,300%	36,400%
$\left[0, 0.75 \times \sum_j p_j\right]$	$[1, 10]$	3860%	58,870%	26%
	$[1, 20]$	2442%	2193%	5149%
	$[1, 30]$	0.3%	583%	117%
$\left[0, \sum_j p_j\right]$	$[1, 10]$	124%	7%	7%
	$[1, 20]$	7%	11%	0.3%
	$[1, 30]$	0.3%	0.6%	5%

As a result, the heuristic approach provides a superior performance and is usually able to reach the optimal solution. Also, for the cases where optimal solution is not gained by this method, the value of *APD* is almost very low.

4.3. Impact of dominance rules on the B&B algorithm

In order to determine the efficiency of applying heuristic algorithm in the B&B algorithm and the *DR*1, we have resolved small instances using this method in the case of not using the results of heuristic algorithm as the initial solutions. Considering the results shown in Table 14, where the numbers stand for the percentage of increase in the number of nodes of the search tree, it is inferred that using the heuristic algorithm will significantly reduce the number of created nodes especially when r_j takes small values. It is also concluded that this approach has a negligible effect on our results in some cases where r_j has a large value, due to the inappropriate lower bound in such instances.

Table 15 reports the influence of the *DR*1 in terms of the number of nodes in the search tree. As is shown, discarding this rule will considerably increase the number of created nodes in all instances.

5. Impact of inventory capacity

To illustrate the impact of change in inventory capacity of the terminal, we have randomly chosen one small-size instance from each combination of the three parameters of n, p_j and r_j (resulting in totally

Table 15. Impact of the DR1 in the B&B algorithm.

r_j	p_j \ n	20	25	30
$\left[0, 0.5 \times \sum_j p_j\right]$	$[1, 10]$	43%	5%	7%
	$[1, 20]$	1%	3%	3%
	$[1, 30]$	1%	1%	112%
$\left[0, 0.75 \times \sum_j p_j\right]$	$[1, 10]$	10%	30%	1301%
	$[1, 20]$	190%	232%	141%
	$[1, 30]$	49%	989%	2%
$\left[0, \sum_j p_j\right]$	$[1, 10]$	252%	207%	10%
	$[1, 20]$	327%	51%	197%
	$[1, 30]$	13%	17%	398%

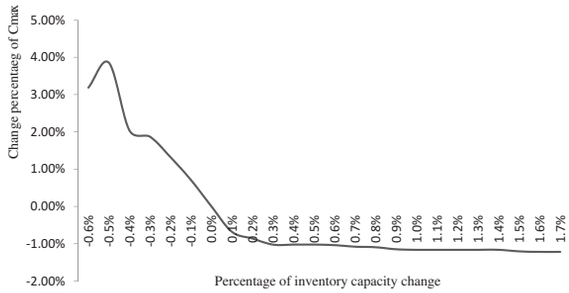

Figure 6. Impact of inventory capacity change on the objective value.

27 instances). In order to conduct sensitivity analyses on I_c, we have increased and decreased this parameter, respectively, until no more changes are observed in the solution and no feasible solution is found. In Figure 6, the horizontal axis shows the percentage of variations in I_c and the vertical one depicts the average change percentage in the objective value. Since in some problems a decrement in I_c would quickly lead to infeasibility of the problem, the results presented for large decreases are only related to the instances with feasible solutions.

The figures indicate that decreasing inventory capacity for more than 0.6% leads to infeasible solution for all of our generated test instances. As given in this figure, more than 1.5% increase in the inventory capacity results in no improvement in the answer. Moreover, any decrease in this parameter leads to at most 4% of increasing in the objective value.

We expect when the inventory capacity decreases, the makespan should increase but there is an exception when capacity decreases from −0.5% to −0.6%. In fact, for each test instance, the makespan increases when the inventory capacity decreases. But the cause of this unusual phenomenon is that when the inventory capacity decreases to −0.6%, for some of the test instances there is not any feasible solution. On the other hand, since this curve indicates the average of change percentage of the makespan and in this case, some of test instances are omitted, the average might decrease.

6. Conclusion

In this paper, we consider a scheduling problem in a single station transshipment terminal where considerations for trucks' release date have been taken into account as well as inventory constraints. This problem was formulated as an integer linear programming model, which its performance was significantly affected by the number of jobs and also the processing times. We proposed two exact approaches, the B&B algorithm and dynamic programming method, to efficiently solve the small-size problems. In addition, we presented an efficient heuristic algorithm for both small and large instances, able to reach optimal solution in most of the problems.

For future research, we identify the following directions to be promising. First, considering more than one type of products seems to be interesting. Second, analyzing the problem scheduling with uncertainty of truck arrival times and processing times can be motivated from real-world problems. Third, we can think of different objectives such as minimizing (1) the maximum lateness, (2) weighted sum of completion times, (3) number of delayed trucks and (4) total weighted tardiness in the cross-dock.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Masoumeh Ghorbanzadeh is a PhD student in the Department of Industrial Engineering at the Ferdowsi University of Mashhad. She received her B.Sc. and M.Sc. degrees in Department of Industrial Engineering at the Ferdowsi University of Mashhad. Her research activities include machine scheduling, logistics and supply chain.

Mohammad Ranjbar is an associate professor of Industrial Engineering in the Department of Industrial Engineering at the Ferdowsi University of Mashhad. He received his B.Sc. and Ph.D. degrees in the Industrial Engineering Department of the Sharif University of Technology and his M.Sc. degree in the Industrial Engineering Department of University of Tehran. His research activities include machine and project scheduling, logistics and supply chain.

Negin Jamili is a PhD student in the Department of Technology and Operations Management at the Rotterdam School of Management, Erasmus University. She received her B.Sc. and M.Sc. degrees in the Department of Industrial Engineering at the Ferdowsi University of Mashhad. Her research activities include machine scheduling, warehousing, logistics and supply chain.

References

[1] Wen M, Larsen J, Clausen J, et al. Vehicle routing with cross-docking. J Oper Res Soc. 2009;60 (12):1708–1718.

[2] Yu W, Egbelu PJ. Scheduling of inbound and outbound trucks in cross docking systems with temporary storage. Eur J Oper Res. 2008;184(1):377–396.

[3] Boysen N, Fliedner M. Cross dock scheduling: classification, literature review and research agenda. Omega (Westport). 2010;38(6):413–422.

[4] Buijs P, Vis IF, Carlo HJ. Synchronization in cross-docking networks: a research classification and framework. Eur J Oper Res. 2014;239(3):593–608.

[5] Chen F, Lee C-Y. Minimizing the makespan in a two-machine cross-docking flow shop problem. Eur J Oper Res. 2009;193(1):59–72.

[6] Arabani AB, Zandieh M, Ghomi SF. Multi-objective genetic-based algorithms for a cross-docking scheduling problem. Appl Soft Comput. 2011;11 (8):4954–4970.

[7] Arabani AB, Zandieh M, Ghomi SF. A cross-docking scheduling problem with sub-population multi-objective algorithms. Int J Adv Manuf Technol. 2012;58(5–8):741–761.

[8] Forouharfard S, Zandieh M. An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems. Int J Adv Manuf Technol. 2010;51(9–12):1179–1193.

[9] Ghobadian E, Tavakkoli-Moghaddam R, Javanshir H, et al. Scheduling trucks in cross docking systems with temporary storage and dock repeat truck holding pattern using GRASP method. Int J Ind Eng Computations. 2012;3(5):777–786.

[10] Vahdani B, Soltani R, Zandieh M. Scheduling the truck holdover recurrent dock cross-dock problem using robust meta-heuristics. Int J Adv Manuf Technol. 2010;46(5–8):769–783.

[11] Vahdani B, Zandieh M. Scheduling trucks in cross-docking systems: robust meta-heuristics. Comput Ind Eng. 2010;58(1):12–24.

[12] Arabani ARB, Ghomi SF, Zandieh M. A multi-criteria cross-docking scheduling with just-in-time approach. Int J Adv Manuf Technol. 2010;49(5–8):741–756.

[13] Alvarez-Perez GA, González-Velarde JL, Fowler JW. Crossdocking—just in Time scheduling: an alternative solution approach. J Oper Res Soc. 2009;60 (4):554–564.

[14] Sadykov R. Scheduling incoming and outgoing trucks at cross docking terminals to minimize the storage cost. Ann Oper Res. 2012;201(1):423–440.

[15] Boysen N. Truck scheduling at zero-inventory cross docking terminals. Comput Oper Res. 2010;37 (1):32–41.

[16] Boysen N, Fliedner M, Scholl A. Scheduling inbound and outbound trucks at cross docking terminals. OR Spectr. 2010;32(1):135–161.

[17] Konur D, Golias MM. Analysis of different approaches to cross-dock truck scheduling with truck arrival time uncertainty. Comput Ind Eng. 2013;65(4):663–672.

[18] Miao Z, Lim A, Ma H. Truck dock assignment problem with operational time constraint within crossdocks. Eur J Oper Res. 2009;192(1):105–115.

[19] Miao Z, Cai S, Xu D. Applying an adaptive tabu search algorithm to optimize truck-dock assignment in the crossdock management system. Expert Syst Appl. 2014;41(1):16–22.

[20] Chen F, Song K. Minimizing makespan in two-stage hybrid cross docking scheduling problem. Comput Oper Res. 2009;36(6):2066–2073.

[21] Van Belle J, Valckenaers P, Berghe GV, et al. A tabu search approach to the truck scheduling problem with multiple docks and time windows. Comput Ind Eng. 2013;66(4):818–826.

[22] Soltani R, Sadjadi SJ. Scheduling trucks in cross-docking systems: A robust meta-heuristics approach. Transp Res E Logist Transp Rev. 2010;46 (5):650–666.

[23] Boysen N, Briskorn D, Tschöke M. Truck scheduling in cross-docking terminals with fixed outbound departures. OR Spectr. 2013;35(2):479–504.

[24] Alpan G, Ladier A-L, Larbi R, et al. Heuristic solutions for transshipment problems in a multiple door cross docking warehouse. Comput Ind Eng. 2011;61 (2):402–408.

[25] Alpan G, Larbi R, Penz B. A bounded dynamic programming approach to schedule operations in a cross docking platform. Comput Ind Eng. 2011;60 (3):385–396.

[26] Maknoon MY, Koné O, Baptiste P. A sequential priority-based heuristic for scheduling material handling in a satellite cross-dock. Comput Ind Eng. 2014;72:43–49.

[27] Mohtashami A. Scheduling trucks in cross docking systems with temporary storage and repetitive pattern for shipping trucks. Appl Soft Comput. 2015;36:468–486.

[28] Shakeri M, Low MYH, Turner SJ, et al. A robust two-phase heuristic algorithm for the truck scheduling problem in a resource-constrained crossdock. Comput Oper Res. 2012;39(11):2564–2577.

[29] Hermel D, Hasheminia H, Adler N, et al. A solution framework for the multi-mode resource-constrained cross-dock scheduling problem. Omega (Westport). 2016;59:157–170.

[30] Briskorn D, Choi B-C, Lee K, et al. Complexity of single machine scheduling subject to nonnegative inventory constraints. Eur J Oper Res. 2010;207(2):605–619.

[31] Briskorn D, Jaehn F, Pesch E. Exact algorithms for inventory constrained scheduling on a single machine. J Scheduling. 2013;16(1):105–115.

[32] Briskorn D, Leung JY. Minimizing maximum lateness of jobs in inventory constrained scheduling. J Oper Res Soc. 2013;64(12):1851–1864.

[33] Briskorn D, Pesch E. Variable very large neighbourhood algorithms for truck sequencing at transshipment terminals. Int J P Res. 2013;51 (23–24):7140–7155.

[34] Bazgosha A, Ranjbar M, Jamili N. Scheduling of loading and unloading operations in a multi stations transshipment terminal with release date and inventory constraints. Comput Ind Eng. 2017;106:20–31.

[35] Graham RL, Lawler EL, Lenstra JK, et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of discrete mathematics. Vol. 5. Elsevier; 1979. p. 287–326.

[36] Davari M, Ranjbar M, Leus R, "A block-based branch-and-bound algorithm for transshipment scheduling problem," Technical Report, KU Leuven, (2018).