Applied Soft Computing Journal 98 (2021) 106736

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

A variable neighborhood search algorithm for transshipment scheduling of multi products at a single station

Mohammad Ranjbar *, Reza Ghorbani Saber

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

ARTICLE INFO

Article history:
Received 27 March 2018
Received in revised form 16 July 2020
Accepted 15 September 2020
Available online 17 September 2020

Keywords:
Scheduling
Transshipment terminal
Variable neighborhood search algorithm

ABSTRACT

In this research, we investigate a transshipment scheduling problem in which a set of loading and unloading jobs must be performed at a single station of a transshipment terminal. We assume that there is a set of product types and an inventory storage center with a constant capacity for each. Each loading job decreases the inventory level of a particular product type and has a predetermined due date, while each unloading job increases the inventory level of a product type and has a given release date. An inbound truck delivers a set of products, which have the same release dates and must be unloaded, to the inventory storage center. In contrast, an outbound truck conveys a set of goods, which have identical completion times and must be loaded. We aim to minimize the total freight cost of trucks as well as the total weighted tardiness of products. To this end, we develop a linear integer programming model and a variable neighborhood search algorithm including a set of efficient local search procedures. Moreover, we run the two previously developed metaheuristics, i.e. a genetic algorithm and a particle swarm algorithm so as to solve the problem. Using a set of randomly generated test instances, we perform extensive computational experiments and statistical analyses to highlight the efficiency and superiority of the developed algorithm.

1. Introduction

In this paper, we consider a transshipment terminal, named a cross-dock, with a single station where different products are received and delivered at given dates using a set of trucks. Each inbound truck carries various types of products, and we consider all as a batch that must be unloaded at the transshipment terminal. Furthermore, an unloading job includes unloading all products of the same type in a batch. For each type of product, there is a limited capacity for temporary storage in the transshipment terminal. Moreover, a demand consists of a particular amount of different products as well as a due date. A loading job includes loading products asked by a demand. Outbound trucks only can be loaded using loading batches, consisting of consolidating several loading jobs. In addition, it is assumed that partial delivers of loading job are not allowed, all trucks have identical capacity and the single loading/unloading station can process only one job at a time. We aim to construct loading batches and find a sequence for loading and unloading jobs in such a way that the total cost of outbound trucks and weighted tardiness is minimized.

The problem at hand is a generalization of the problem introduced by Briskorn and Leung [1], in which a single type of product was considered, and inventory capacity of the transshipment terminal was supposed to be unlimited (shown as $1|inv|L_{max}$ using the notation of Graham et al. [2]). Having aimed to minimize the maximum lateness, they proved that the problem is NP-hard and developed a set of branch-and-bound algorithms. Briskorn et al. [3] studied the complexity of a set of single-machine scheduling problems in which nonnegative inventory constraints were factored in. Bazgosha et al. [4] investigated $Pm|inv|C_{max}$ with a single type of product and limited capacity for the inventory storage center. They developed two scheduling generation schemes, i.e. serial and parallel, and also developed three metaheuristic algorithms based on genetic algorithm, particle swarm optimization and cuckoo optimization algorithms.

Rarely are papers in the field of scheduling with inventory constraints consideration. We can cite Schwindt and Trautmann [5], Neumann and Schwindt [6], Neumann et al. [7] and Bartels and Zimmermann [8] as articles that addressed this issue. Buijs et al. [9] and Boysen and Fliedner [10] seem to be the two most recent related surveys that studied cross-docking systems. In this context, most research papers intended to find a sequence of inbound and outbound trucks to minimize operational costs. Boysen et al. [11] studied the scheduling of trucks at cross-dock

* Corresponding author.
E-mail address: m_ranjbar@um.ac.ir (M. Ranjbar).

https://doi.org/10.1016/j.asoc.2020.106736

terminals where trucks are capable of both picking and delivering products. They also assumed that the terminal has two separate stations for the process of loading and unloading trucks. Arabani et al. [12] analyzed a multi-objective cross-docking problem in which items are unloaded in a single receiving gate and loaded onto one available shipping gate. Moreover, defining a scheduling problem at a transshipment terminal where products should be picked up by trucks immediately after unloading, Boysen [13] developed a dynamic programming algorithm in conjunction with a metaheuristic approach based on the simulated annealing algorithm. Yu and Egbelu [14] assumed a temporary storage area at the front of the shipping dock in order to store the finished products shortly. As a result, some of these products may wait in the temporary storage area until an appropriate outbound truck becomes available. Forouharfard and Zandieh [15] also studied a similar problem and developed an imperialist competitive algorithm (ICA). Considering no temporary storage area, Vahdani et al. [16] let the trucks to return to their corresponding dock in order to end their task.

The contributions of this paper are twofold: (1) we develop a mixed linear integer programming model for the introduced problem and solve it using the ILOG CPLEX; and (2) we develop a metaheuristic algorithm based on the variable neighborhood search (VNS) algorithm which benefits from efficient moves in its local search procedure. To evaluate the performance of our developed VNS, we modify the genetic algorithm (GA) and particle swarm optimization (PSO) algorithm developed by Bazgosh et al. [4] and present extensive comparative results.

The remainder of this paper is organized as follows. In Section 2, problem description and formulation as a linear integer programming model is presented. In Section 3, a VNS algorithm is developed and GA and PSO algorithms of Bazgosh et al. [4] are modified to solve our problem. The developed model and algorithms are evaluated using extensive comparative computational experiments in Section 4. Finally, conclusions are drawn, and future research directions are discussed in Section 5.

2. Problem description and modeling

We consider a transshipment scheduling problem in which a set of jobs J must be processed at a single station that can process only one job at a time. Each job should be either loaded or unloaded at the station. We suppose that there is unlimited number of trucks, and each has a fixed capacity V and a constant cost F. There are two types of jobs, i.e. unloading jobs $J^+ = \left\{j_1^+, \ldots, j_{|J^+|}^+\right\}$ and loading jobs $J^- = \left\{j_1^-, \ldots, j_{|J^-|}^-\right\}$. We also assume that there are pt types of products and K_k; $k = 1, \ldots, pt$ indicates the set of (unloading and loading) job type k where $\bigcup_{k=1}^{pt} K_k = J$. Each inbound truck contains an unloading batch B_b. Each batch includes a given set of unloading jobs that each consists of a predetermined number of a particular product type. Furthermore, each outbound truck contains a loading batch. A loading batch includes a set of loading jobs that each consists of a given number of a product type. We show all unloading and loading batches by means of $B^+ = \left\{B_1^+, \ldots, B_{|B^+|}^+\right\}$ ($B^+ = J^+$) and $B^- = \left\{B_1^-, \ldots, B_{|J^-|}^-\right\}$ ($B^- = J^-$), respectively. Each job j is associated with a processing time p_j and an inventory modification δ_{jk}. For each product type k, the initial inventory level I_k^{ini} and the inventory capacity I_k^C, specified by a number of item unit, are given. We assume that preemption of jobs is not allowed, and jobs of an unloading batch should be processed consecutively. In other words, processing of an unloading batch cannot be interrupted with a loading job, whereas the processing of a loading batch may be interrupted by an unloading batch. In addition to this, for each unloading batch B_b, we are given a release date r_b indicating the release date of all including jobs. Moreover, requiring a capacity v_j to be stored in a truck, a loading job j has a due date d_j and a delay cost w_j calculated per time unit. Each loading job must be delivered to a customer. We assume that all customers are placed in the vicinity of the transshipment terminal such that we can dismiss the delivering times.

In the developed model, first, we have to decide which loading jobs should be included in a loading batch b; $b = 1, \ldots, |J^-|$. Secondly, we must determine a schedule establishing the sequence of unloading and loading batches so as to minimize the cost of trucks and delays of loading jobs. This should be noted that the maximum number of loading batches is $|J^-|$ and each loading batch includes only one job. We also consider a limited planning horizon as $T = \{1, \ldots, |T|\}$ where $|T| = \max_{b=1,\ldots,|B^+|}\{r_b\} + \sum_{j\in J} p_j$.

The following decision variables are utilized to model the described problem.

$$XL_{jt} = \begin{cases} 1; \text{ If loading job } j \text{ finishes at time } t \\ 0; \text{ Otherwise} \end{cases}$$

$$XU_{bt} = \begin{cases} 1; \text{ If unloading batch } b \text{ finishes at time } t \\ 0; \text{ Otherwise} \end{cases}$$

$$Y_{jb} = \begin{cases} 1; \text{ If loading job } j \text{ is placed in batch } b \\ 0; \text{ Otherwise} \end{cases}$$

$$Z_b = \begin{cases} 1; \text{ If loading batch } b \text{ is not empty} \\ 0; \text{ Otherwise} \end{cases}$$

$C_b =$ Completion time of loading batch$b$

$T_j =$ Tardiness of loading job j

$I_{kt} =$ Inventory level of product type k at time t

$M =$ abig number

We present the mathematical formulation of the problem as follows.

$$Min\ F\sum_{b=1}^{|J^-|} Z_b + \sum_{j=1}^{|J^-|} w_j T_j \quad (1)$$

Subject to:

$$\sum_{\forall t\in T} XL_{jt} = 1 \qquad \forall j \in J^- \quad (2)$$

$$\sum_{\forall t\in T} XU_{jb} = 1 \qquad \forall b: B_b^+ \subseteq B^+ \quad (3)$$

$$\sum_{\forall j\in J^-} XL_{jt} + \sum_{\forall b: B_b^+ \subseteq B^+} XU_{bt} \le 1 \qquad \forall t \in T \quad (4)$$

$$\sum_{\tau=t-p_j+1}^{t} \sum_{j'\in J^-\setminus j} XL_{j'\tau} \le M\left(1 - XL_{jt}\right) \qquad \forall j \in J^-; \forall t \in T \quad (5)$$

$$\sum_{\tau=t-p_j+1}^{t} \sum_{B_b^+ \subseteq B^+} XU_{b\tau} \le M\left(1 - XL_{jt}\right) \qquad \forall j \in J^-; \forall t \in T \quad (6)$$

$$\sum_{\tau=t-\sum_{j\in B_b^+} p_j+1}^{t} \sum_{B_{b'}^+ \subseteq B^+\ b'\neq b} XU_{b'\tau} \le M\left(1 - XU_{bt}\right) \qquad \forall b: B_b^+ \subseteq B^+; \forall t \in T \quad (7)$$

$$\sum_{\tau=t-\sum_{j\in B_b^+} p_j+1}^{t} \sum_{j\in J^-} XL_{j\tau} \le M\left(1 - XU_{bt}\right) \qquad \forall b: B_b^+ \subseteq B^+; \forall t \in T \quad (8)$$

$$\sum_{t\in T} tXL_{jt} \ge p_j \qquad \forall j \in J^- \quad (9)$$

$$\sum_{t\in T} tXU_{bt} - r_b \ge \sum_{j\in B_b^+} p_j \qquad \forall b: B_b^+ \subseteq B^+ \quad (10)$$

$$Z_b \ge \frac{\sum_{j\in B_b^-} Y_{jb}}{M} \qquad \forall b: B_b^- \subseteq B^- \quad (11)$$

$$C_b \geq tXL_{jt} + M\left(Y_{jb} - 1\right) \quad \forall j \in J^-; \forall t \in T; \forall b: B_b^- \subseteq B^- \quad (12)$$

$$T_b \geq C_b - d_j + M\left(Y_{jb} - 1\right) \quad \forall j \in J^-; \forall b: B_b^- \subseteq B^- \quad (13)$$

$$\sum_{j \in J^-} v_j Y_{jb} \leq V \quad \forall b: B_b^- \subseteq B^- \quad (14)$$

$$I_{k0} = I_k^{ini} \quad k = 1, \ldots, pt \quad (15)$$

$$I_{kt} = I_{k,(t-1)} + \sum_{j \in (K_k \cap J^-)} \delta_{jk} XL_{jt} + \sum_{\forall b: B_b^+ \subseteq B^+} \sum_{j \in (K_k \cap B_b^+)} \delta_{jk} XU_{bt} \quad \forall t \in T; k = 1, \ldots, pt \quad (16)$$

$$I_{kt} \leq I_k^C \quad \forall t \in T; k = 1, \ldots, pt \quad (17)$$

$$XU_{jb} \in \{0, 1\} \quad \forall j \in J^+; \forall b: B_b^+ \subseteq B^+ \quad (18)$$

$$XL_{jt}, Y_{jb}, Z_b \in \{0, 1\} \quad \forall j \in J^- \quad \forall t \in T; \forall b: B_b^- \subseteq B^-$$

$$C_b, T_j, I_{kt} \in \mathbb{Z}^+ \quad \forall j \in J^-; \forall t \in T; \forall b: B_b^- \subseteq B^-; k = 1, \ldots, pt \quad (19)$$

The objective function (1) minimizes the total fixed cost of trucks and tardiness cost of loading jobs. Constraints (2) ensure that each job finishes at only one time slot, and Constraints (3) guarantee this issue for each unloading batch. Constraints (4) confirm that at most one job can be completed at any time slot. Constraints (5)–(8) assure that if a loading job or an unloading batch finishes at time slot t, another job cannot be processed during that. In other words, these constraints consider the single resource of the problem as a machine and preclude from preemption of loading jobs and unloading batches. Constraints (9) explain that completion time of a loading job is not less than its processing time and Constraints (10) guarantee that start time of an unloading batch must be equal or greater than its release date. Number of loading batches is calculated using $\sum_{b=1}^{|J^-|} Z_b$ and non-empty loading batches are determined with Constraints (11). Finish times and tardiness values of loading jobs are specified using Constraints (12) and (13), respectively. Constraints (14) impose the capacity constraint of trucks for loading batches. The initial inventory level of each product type is given using Constraints (15), and inventory level of each product type at other time units $t > 0$ are determined through Constraints (16). Constraints (17) consider the inventory capacity constraint for each product type. Finally, the last two sets of constraints describe the type of variables in which $\mathbb{Z}^+$ represents the set of non-negative integers.

3. A VNS algorithm

The problem at hand is NP-hard because it is a generalization of $1|inv|L_{max}$ which has been proved to be NP-hard by Briskorn et al. [3]. Since our developed model, described in Section, is not capable of solving large-sized instances in a reasonable time, designing metaheuristic solution approaches is highly recommended. In this section, we develop a VNS algorithm to find high-quality solutions for the problem. VNS is a metaheuristic algorithm that changes the size and type of neighborhood structure during the search process in a systematic fashion so as to escape from local optima. In the following, we show the solution representation and our VNS description. To evaluate comparatively the performance of our develop VNS, we modify the GA and PSO metaheuristics proposed by Bazgosh et al. [4] to solve our problem.

3.1. Solution representation

We represent a solution s of the problem utilizing a double list $s = \{\sigma, \beta\}$. The list $\sigma = \left(\sigma_1, \ldots, \sigma_{|J^-|+|B^+|}\right)$ indicates a sequence of unloading batches and loading jobs where $\sigma_j \in J^- \cup B^+$. Moreover, the list $\beta = \left(\beta_1, \ldots, \beta_{|J^-|}\right)$ shows batch numbers of loading jobs where β_1 represents batch number of the loading job j_1^-. Different permutations of the first list result in various sequences of job processing at the single station, and different measures of each element in the second list classify loading jobs into different groups. This should be noted that the maxim number of loading batches, which is $|J^-|$, is obtained when each loading job is placed individually in a separate batch. Furthermore, we assume that all unloading jobs of a batch must be processed continuously. In other words, we deal with unloading batches instead of unloading jobs.

As an example, consider an instance including two unloading batches where $B_1^+ = \left\{j_1^+, j_2^+\right\}$ and $B_2^+ = \left\{j_3^+\right\}$ and four loading jobs $\left\{j_1^-, \ldots, j_4^-\right\}$. Apart from inventory constraints, the solution $s = \left\{\left(j_1^-, B_1^+, j_4^-, B_2^+, j_3^-, j_2^-\right), (1, 2, 2, 1)\right\}$ indicates that job j_1^- is processed first and then jobs of batch B_1^+ are unloaded (in an arbitrary sequence). Subsequently, j_4^- is loaded, next batch B_2^+ is unloaded and then jobs j_3^- and j_2^- are loaded. The last four numbers in the solution s show that jobs j_1^- and j_4^- are grouped into a batch B_1^- and are delivered at the completion time of j_4^-. Also, two jobs j_2^- and j_3^- are loaded simultaneously as batch B_2^- and are delivered as soon as j_2^- finishes.

3.2. Sketch of the VNS

VNS, proposed by Mladenović and Hansen [17], is a metaheuristic method for solving a set of combinatorial optimization problems. It explores distant neighborhoods of the current solution and moves to a new solution if and only if an improvement is made. The first step of a VNS is definition of solution representation. Afterward, neighborhood structures are designed on the basis of the proposed solution representation. Our developed VNS, presented in Algorithm 1, includes three main procedures, i.e. shaking, local search and repairing procedures.

Algorithm 1: Sketch of the VNS

```
Input: Sh_max and TL
  s ← generate a random initial solution
  while current-time < TL do
      sh ← 1;
      while sh ≤ sh_max do
          s' ← Shake (s, sh);
          s'' ← VND (s');
          if f(s'') < f(s) do
              s ← s'';
              sh ←1;
          Else
              sh ← sh + 1;
          End
      End
  End
```

Shaking procure is designed to avoid getting trapped in local optima and is performed using two efficient moves, i.e. λ-reverse and γ-insert. The local search procedure aims to intensify the search process in some areas of the search space of the problem. We choose a variable neighborhood descent (VND) as the local search algorithm. This explores systematically several neighborhood structures using λ-opt, λ-reverse, γ-swap and γ-insert moves, which are described in Section 3.5. Besides, we develop a repairing procedure, described in Section 3.6, to convert each infeasible solution to a feasible one.

Our developed VNS gets the two following parameters as inputs: the maximum number of moves in the shaking procedure (Sh_{max}) and the time limit of the algorithm (*TL*). Next, an initial solution s is generated randomly. Since this solution might be infeasible, it has to be repaired with the repairing procedure. Afterward, a *While* loop is performed until the given time limit is met. At each iteration of the VNS algorithm, a shaking procedure is first applied to the current solution and the local search procedure is performed afterward.

3.3. Neighborhood structures

To construct a set of neighbors for a solution, we need to introduce different kinds of moves for two lists σ and β. For a list σ, we consider two types of moves, named λ-opt and λ-*reverse*. The λ-*opt* move extracts λ; $\lambda = 1, \ldots, \left|J^-\right| + \left|B^+\right| - 1$ consecutive elements from the current list σ and inserts them in a new random position. Let us consider the example of Section 3.1 where $\sigma = \left(j_1^-, B_1^+, j_4^-, B_2^+, j_3^-, j_2^-\right)$. If we assume that $\lambda = 2$ and elements of B_1^+ in conjunction with j_4^- are inserted immediately after j_3^-, the new list $\sigma' = \left(j_1^-, B_2^+, j_3^-, B_1^+, j_4^-, j_2^-\right)$ is obtained. On the other hand, the λ-reverse move selects λ; $\lambda = 1, \ldots, \left|J^-\right| + \left|B^+\right| - 1$ consecutive elements from the current list σ and reverses their relative order. If $\lambda = 2$ and we are going to select again elements B_1^+ along with j_4^-, the new list $\sigma'' = \left(j_1^-, j_4^-, B_1^+, B_2^+, j_3^-, j_2^-\right)$ is attained. If the newly obtained list σ is infeasible because of inventory constraints violation, it will be repaired using the repairing procedure. Besides, for each list β, we consider two sorts of move named γ-*swap* and γ-*insert* where $\gamma = 1, \ldots, \frac{|J^-|}{2}$. The 1-*swap* move selects one pair of loading jobs casually, and if capacity and inventory constraints are not violated, it exchanges their batch numbers. The γ-*swap* is a random replication of 1-*swap* for γ times. For instance, if we let $\beta = (1, 2, 2, 1)$ in the example, swap batch numbers of jobs j_1^- and j_2^- and ignore the truck capacity constraints, the new list $\beta' = (2, 1, 2, 1)$ is acquired. Furthermore, the 1-*insert* move selects a loading job and inserts it in a new batch if both inventory and capacity constraints are satisfied. If we consider $\beta = (1, 2, 2, 1)$ and insert job j_2^- into the first batch, the new list $\beta'' = (1, 1, 2, 1)$ is attained. In other words, the γ-*insert* move is a random replication of 1-*insert* for γ-times.

3.4. A shaking procedure

Shaking procedure is proposed to escape from local optimal solutions and is a diversification strategy that is utilized when no improvement is reached. This procedure uses parameter *sh* as the number of shaking operations applied to lists σ and β. In other words, it performs *sh* times 2-*reverse* and 2-*insert* moves to lists σ and β, respectively.

3.5. A local search procedure

Our local search procedure is a variable neighborhood descent (VND) procedure that searches in the developed neighborhood structures based upon the first improvement strategy. This procedure completely explores one neighborhood structure and extends its search to the next neighborhood structure in the case of reaching no better solution (Todosijević et al. [18]).

The general structure of our developed VND is shown in Algorithm 2.

Algorithm 2: The VND procedure

Input: s (a solution), $\lambda^{max} = |J^-| + |B^+| - 1$, $\gamma^{max} = {|J^-|}/{2}$;
$\lambda \leftarrow 1$; $\gamma \leftarrow 1$; *imp*=false;
while ($\lambda \le \lambda^{max}$ & $imp = false$) **do**
  $s' \leftarrow \lambda - swap\ (\lambda, s)$;
  If s' is infeasible, $s' \leftarrow repair\ (s')$;
  if $f(s') < f(s)$ **do**
    $s \leftarrow s'$;
    imp= true;
  else
    $s' \leftarrow \lambda - insert\ (\lambda, s)$;
    If s' is infeasible, $s' \leftarrow repair\ (s')$;
    If $(s') < f(s)$ **do**
      $s \leftarrow s'$;
      imp= true;
    End
  End
  If $(imp = false)$
    $\lambda = \lambda + 1$;
End
imp=false;
while ($\gamma \le \gamma^{max}$ & $imp = false$) **do**
  $s' \leftarrow \gamma - swap\ (\lambda, s)$;
  If s' is infeasible, $s' \leftarrow repair\ (s')$;
  if $f(s') < f(s)$ **do**
    $s \leftarrow s'$;
    imp= true;
  else
    $s' \leftarrow \gamma - insert\ (\lambda, s)$;
    If s' is infeasible, $s' \leftarrow repair\ (s')$;
    If $(s') < f(s)$ **do**
      $s \leftarrow s'$;
      imp= true;
    End
  End
  If $(imp = false)$
    $\gamma = \gamma + 1$;
End

In this procedure, the three following parameters are considered as inputs: maximum number of λ ($\lambda^{max} = \left|J^-\right| + \left|B^+\right| - 1$), the maximum number of γ ($\gamma^{max} = \frac{|J^-|}{2}$) and binary variable *imp* which indicates whether improvement has been obtained (*imp* = true) or not (*imp* = false). It replaces the current solution s with a new better one and restarts from the first neighborhood structure when an improvement is achieved. In other words, the VND procedure starts with $\lambda = \gamma = 1$ and applies the $\lambda - opt$ move. If no improvement is found, then the $\lambda - reverse$ move is applied. Subsequently, the γ-swap and γ-insert are performed, respectively. In the next step, if the current solution is not improved, the VND procedure initializes the parameters λ and γ as $\lambda = \gamma = 2$ and then implements the $\lambda - opt$, $\lambda - reverse$, γ-swap and γ-insert moves alternatively until an improvement is attained. This procedure is stopped whenever no improvement is found in all of the neighborhood structures. In each step of the local search procedure we may find an infeasible that must be converted to a feasible one using the repairing procedure.

3.6. A repairing procedure

This procedure takes an infeasible solution and transforms it into a feasible one. An infeasible solution to the problem at hand may have different reasons for infeasibility, where one of them is violation of the inventory constraint. In this case, if an unloading batch cannot be unloaded due to violation of the inventory capacity for some product types, this batch is shifted to the right side in the list σ. We consider the minimum number of shifts such that some loading jobs from those product types are placed before this batch and inventory capacity is satisfied. In addition to this, if a loading job cannot be processed because of inventory shortage, this job is shifted to the right side in the list σ. We choose the minimum number of shifts in such a way that unloading batches which include enough inventory of the product type encountered shortage are placed before that job. This should be noted that each infeasible list σ is repaired from the left side to the right side.

A list β might be infeasible because a loading batch may violate the truck capacity. In this case, we remove the minimum number of jobs with the smallest values of v_j from that batch so that the truck capacity constraint is satisfied. This is worth mentioning that loading batch numbers should be consecutive and if this condition is not held in a solution, we must renumber all batches.

3.7. Modified GA and PSO

In this section, we modify the GA and PSO solution approaches developed by Bazgosha et al. [4] to perform them for our problem. We chose these two algorithms because they have been developed for a loading and unloading scheduling problem in a multi-station transshipment terminal. To do so, we need to modify these two algorithms due to the following reasons.

(I) They considered only one product type instead of several types
(II) They considered multi docks instead of one station
(III) They did not consider any due date for the loading jobs
(IV) Their objective function is different from ours
(V) They did not define batches as we do

GA is a population-based metaheuristic developed by Holland [19] including two main operators called cross-over and mutation. This solution approach has been used widely to solve complex optimization problems in the field of cross-docking scheduling (See Boloori et al. [20], Molavi et al. [21] and Tamannaeib et al. [22]). Following Bazgosh et al. [4], we employed the well-known two-point crossover operator and a permutation operator, described by them. The crossover operator is applied to all newly generated solutions while permutation operator is used with a probability of p_{mut}. Moreover, the generated infeasible solutions are repaired using our developed repairing procedure. All other settings are copied from Bazgosha et al. [4] and only our solution representation and objective function are different from theirs. This algorithm includes parameters p_{mut} and $|Pop|$ (population size) which will be tuned in Section 4.1.

PSO is also a population-based search algorithm developed by Eberhart and Kennedy [23], inspired by the flocking behavior of birds. There are several papers that have utilized PSO algorithm to solve cross-docking scheduling problems (See Mohtashami and Tavana [24] and Wisittipanich and Hengmeechai [25]). For PSO, we choose to work with our developed solution representation instead of binary representation. We implement Algorithm 4 of Bazgosha et al. [4] as our PSO approach and just replace our objective function instead of theirs. Moreover, we use the same velocity equations but we apply it to both lists σ and β. Besides, we convert new generated infeasible solutions to infeasible ones using the repairing procedure. This algorithm includes parameters c_1, c_2, V_{max}(maximum velocity) and $|Pop|$ (population size) which will be tuned in Section 4.1.

4. Computational experiments

We coded all developed algorithms including the VNS, GA and PSO in Visual C++ 2015 and performed all computational results on a computer with an Intel Core i5 and 8GB of RAM. In addition, we used the IBM CPLEX 12.7.1 as a solver for the linear integer programming model.

4.1. Test set generation and parameters setting

We generated instances with $n = 10, 20, 30, 40$ and 50 jobs and $pt = 2, 3$ and 4 product types. Since the processing times may influence the performance of the developed model, we chose them randomly from the discrete uniform distribution $\{1, \ldots, \pi\}$ where $\pi \in \{10, 30\}$. The integer release dates were chosen from the discrete uniform distribution $\{1, \ldots, \rho \sum_{j\in J} p_j\}$ where $\rho \in \{1, 1.5, 2\}$ and inventory modifications were drawn randomly from the discrete uniform distribution $\{1, \ldots, 10\}$. We considered $V = 10$ for all inbound and outbound trucks and $v_j; \forall j \in J^-$ is an integer quantity randomly taken from discrete uniform distribution $1, \ldots, 5$. We supposed that all inbound trucks are filled as many as possible and each may include various types of products. Moreover, we assumed that $F = 500$ cost unit and $w_j; \forall j \in J^-$ is an integer value randomly selected from the discrete uniform distribution {200,…,400}. The processing time of each job $\left(p_j\right)$ and the release date of each batch (r_b) were taken from the discrete uniform distributions $\{1, \ldots, 10\}$ and $\{0, \ldots, \sum_{\forall j} p_j\}$, respectively.

Jobs were assigned to sets $J = J^+ \cup J^-$ and $K_k; k = 1, \ldots, pt$ with equal probabilities. Finally, we generated amounts of I_k^{ini} and I_k^C for $k = 1, \ldots, pt$ randomly from the discrete uniform distributions $\left\{\max\left(0, -\sum_{j\in K_k} \delta_j\right), \ldots, -\sum_{j\in(K_k\cap J^-)} \delta_j\right\}$ and $\left\{I_k^{ini} + \max\left(0, \sum_{j\in K_k} \delta_j\right), \ldots, I_k^{ini} + \sum_{j\in(K_k\cap J^+)} \delta_j\right\}$, respectively. For each combination of parameters n, pt, π and ρ, we generated 10 random instances resulting in 900 test instances.

We also set parameter sh of VNS algorithm using fine-tuning as $sh = \frac{|J^-|+|B^+|}{2}$ and presents the results of VNS for time limits $TL = 1, 10, 30, 60$ and 180 s. Likewise, we set parameters of the GA as $p_{mut} = 0.05$ and $|pop| = 150$ and parameters of the PSO as $V_{max} = 4, c_1 = c_2 = 2$ and $|pop| = 150$.

4.2. Comparative results

In this section, we compare the results of the developed model, VNS algorithm, GA and PSO algorithm. Table 1 indicates the average runtimes with $TL = 3600$ seconds as well as the number of problems solved optimally using the CPLEX, shown inside parentheses. Since the runtimes of the model for the test instances with $n > 30$ is intractable, this model was not implemented for those.

The results of Table 1 indicate that the runtime of our developed model is highly dependent on the parameter p_j. The larger the size of an instance is, the more runtime the instance needs. This is due to the fact that the variables of the model are directly dependent on the time indices. A similar trend is also observed for the parameter r_j. Furthermore, the runtimes seem to be increased by escalation the parameters n and pt. However, 485 out of 540 test instances, which is around 90%, for $n \leq 30$ were optimally solved by the CPLEX. We call this set of instances as Set1 and

Table 1
The average runtimes (in seconds) of the developed model.

	$\rho = 1$, $\pi = 10$	$\rho = 1$, $\pi = 30$	$\rho = 1.5$, $\pi = 10$	$\rho = 1.5$, $\pi = 30$	$\rho = 2$, $\pi = 10$	$\rho = 2$, $\pi = 30$
$n = 10$, pt=2	8.1(10)	218.9(10)	6.3(10)	114.8(10)	5.8(10)	109.6(10)
$n = 10$, pt=3	11.3(10)	293.3(10)	7.5(10)	200.8(10)	7.0(10)	168.8(10)
$n = 10$, pt=4	18.4(10)	354.7(10)	12.2(10)	288.4(10)	11.9(10)	212.5(10)
$n = 20$, pt=2	20.1(10)	318.9(10)	15.3(10)	224.6(10)	12.8(10)	1194.7(9)
$n = 20$, pt=3	29.7(10)	427.7(10)	21.1(10)	257.1(10)	20.2(10)	1631.2(9)
$n = 20$, pt=4	43.0(10)	561.1(10)	27.0(10)	39.4(10)	27.3(10)	2847.7(8)
$n = 30$, pt=2	192.5(10)	2347.7(10)	314.5(10)	3327.8(8)	547.9(10)	3489.1(8)
$n = 30$, pt=3	144.6(10)	3413.0(9)	397.9(10)	3527.4(7)	742.8(9)	3542.9(8)
$n = 30$, pt=4	185.8(10)	3584.4(9)	455.5(10)	3591.0(6)	1413.1(8)	3595.5(7)

Table 2
Comparative results based on APD.

TL (seconds)	1			10			30			60			180		
n, *pt*	Algorithm														
	GA	PSO	VNS	GA	PSO	VNS	GA	PSO	VNS	GA	PSO	VNS	GA	PSO	VNS
$n = 10, pt = 2$	2.5	2.0	2.3	1.7	1.4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$n = 10, pt = 3$	4.1	3.5	3.7	1.8	1.5	1.2	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$n = 10, pt = 4$	4.7	4.5	4.5	2.2	2.2	1.8	0.9	0.3	0.1	0.2	0.0	0.0	0.0	0.0	0.0
$n = 20, pt = 2$	14.1	13.4	12.8	8.8	7.6	8.6	3.5	3.0	3.4	1.6	1.4	1.5	0.0	0.0	0.0
$n = 20, pt = 3$	16.6	16.8	15.6	9.5	9.6	9.9	4.7	4.8	5.1	2.1	1.8	1.7	0.3	0.0	0.0
$n = 20, pt = 4$	18.6	18.7	18.9	10.7	10.5	10.8	7.8	7.3	7.0	2.7	2.2	1.8	0.1	0.1	0.0
$n = 30, pt = 2$	19.6	19.5	19.4	12.1	11.9	11.7	9.0	8.5	8.1	8.3	7.3	6.5	6.5	6.2	6.3
$n = 30, pt = 3$	22.4	21.8	21.2	14.3	13.8	13.1	10.1	9.3	8.9	9.5	8.8	7.5	7.3	7.4	7.1
$n = 30, pt = 4$	26.1	25.7	24.3	18.0	17.8	18.6	14.1	13.4	12.7	10.5	10.3	9.2	9.8	9.1	8.5
$n = 40, pt = 2$	38.8	37.1	35.8	34.0	33.1	32.6	31.8	30.4	28.4	32.6	30.4	27.3	29.2	28.0	26.9
$n = 40, pt = 3$	42.3	39.7	36.7	35.9	35.0	33.4	33.0	31.7	29.3	33.7	31.3	28.2	32.0	30.0	27.0
$n = 40, pt = 4$	44.2	42.5	40.1	39.9	38.5	36.9	36.4	35.7	33.8	34.6	33.0	31.8	33.8	32.3	30.1
$n = 50, pt = 2$	48.5	46.0	41.4	42.5	40.0	38.4	38.1	36.3	34.3	36.1	33.9	32.5	35.5	33.4	32.1
$n = 50, pt = 3$	51.3	48.4	43.2	47.3	42.4	39.3	40.5	38.3	35.6	38.0	35.3	33.7	36.9	34.6	33.4
$n = 50, pt = 4$	53.7	51.2	44.9	49.8	45.7	41.0	44.0	41.2	38.7	40.1	37.1	35.4	38.0	35.8	33.9
Average APD	**27.2**	**26.1**	**24.3**	**21.9**	**20.7**	**19.9**	**18.3**	**17.4**	**16.4**	**16.7**	**15.5**	**14.5**	**15.3**	**14.5**	**13.7**

address the other 415 instances by Set2. The average runtimes for test instances with $n = 10$, 20 and 30 are 113.9, 428.8 and 1934 s, respectively. Furthermore, the average runtimes for test instances with $pt = 2, 3$ and 4 are 629.7, 824.7 and 959.4 s, respectively. Considering the runtimes trend, we conclude that the number of activities seems has more impact on them rather than the number of product types.

Table 2 indicates the comparative results of three metaheuristic algorithms, i.e. the VNS, the GA and the PSO algorithm. In this table, the results of Set1 are shown based on the average percent deviation (APD) where $\mathrm{PD} = \frac{\text{obtained solution by the algorithm}-\text{best}-\text{found solution}}{\text{best}-\text{found solution}} \times 100$ indicates the percent deviation. In this equation, for each algorithm, the difference between the obtained solution by the algorithm and the best-found solution is calculated and then divided by the best-found solution. The latter is the optimal solution of each instance of Set1, while it is the best-found solution by all developed solution approaches for each instance of Set2. Moreover, we show APD for Set2 based on the difference between the obtained solution by metaheuristic algorithms and the solution found from relaxed version of the mathematical model. The given results highlight the efficiency of our developed VNS such that it is able to find optimal solutions for all instances of Set1 in less than 180 s. In addition to this, it could find optimal solutions for nearly 95% of Set1 within 60 s. In majority of combinations of n and pt, we observe that the VNS performs better than the GA and PSO algorithm. Besides this, these results indicate that the PSO algorithm has better performance than the GA.

4.3. Statistical analyses

To analyze statistically the comparative performance of each pair of developed metaheuristics, we utilize the non-parametric Mann–Whitney U test. To this end, we consider the APD of each algorithm obtained based on each combination of n and pt as a sample. For each statistical test, the null hypothesis is that there is no significant difference between performances of the two considered algorithms. Table 3 shows the U-values of Mann–Whitney U test for each paired comparison in each time limit. Since the sample size is 15 for each algorithm, the critical U-values are 64 and 51 for $\alpha = 0.05$ and 0.01, respectively (α indicates the value of type-I error). If U-value is less than the critical U-value, the null hypothesis is rejected. As can be seen, based upon $\alpha = 0.05$ and 0.01, there are significant differences between performances of algorithms where the VNS shows the best performance while the GA has the poorest one.

Table 3
U-values of Mann–Whitney U test.

Comparison	*TL*(seconds)				
	1	10	30	60	180
VNS vs GA	28	36	21	18	17
VNS vs PSO	48	45	18	17	14
GA vs PSO	22	16	14	14	17

One of the most important factors which highly likely influences the performance of metaheuristic algorithms is the seed of random number generator. Some algorithms are very sensitive to random numbers while some others are almost robust. Therefore, in our experiment, we have selected randomly 10 different instances, namely five instances from Set1 and others from Set2. We perform 30 independent runs in such a way that each run has a unique seed. We measure the runtimes of each algorithm spent solving each instance until the optimal or best-found solution is not reached. The average runtimes (Avg) and their standard deviations (Stdev) are reported in Table 4. In order to analyze these numerical results statistically, we again use the Mann–Whitney U test to compare two algorithms based on

Table 4
Average and standard deviation of runtimes.

Algorithm		Instance #									
		1	2	3	4	5	6	7	8	9	10
VNS	Avg	17.68	18.72	65.17	70.88	118.42	131.48	131.61	164.54	167.07	177.27
	Stdev	1.55	1.25	9.42	10.55	41.44	29.01	32.76	24.45	24.71	20.23
PSO	Avg	18.46	19.95	71.25	78.31	124.58	137.17	147.91	167.36	189.16	196.31
	Stdev	1.56	1.11	8.43	11.04	35.51	32.58	32.12	30.58	22.68	19.75
GA	Avg	20.31	21.11	72.61	82.66	115.13	146.97	160.17	187.12	184.07	227.93
	Stdev	1.24	1.06	8.32	10.00	25.04	24.14	28.69	31.16	25.91	24.30

Table 5
Z-values corresponding to Mann–Whitney U tests.

Comparison	Instance #									
	1	2	3	4	5	6	7	8	9	10
VNS vs GA	5.90	6.12	2.77	4.83	3.25	2.59	3.14	3.06	2.76	5.43
VNS vs PSO	3.62	2.17	2.54	1.67	2.23	1.39	1.18	2.08	1.72	4.13
GA vs PSO	4.30	5.39	0.13	1.05	3.07	1.27	2.08	1.31	0.84	4.54

each instance. Since there are 30 replications (samples) for each algorithm in each test and the critical U-values are available up to 20 samples, we use the corresponding Z-values. It is worth mentioning that the critical Z-values for $\alpha = 0.05$ and 0.01 are 1.95 and 2.58, respectively. According to Table 5 in which the Z-values are shown, VNS beats GA significantly in all instances. Likewise, VNS outperforms PSO in 5 out of 10 instances when $\alpha = 0.05$, but it is not apparent for $\alpha = 0.01$. Furthermore, in nearly half instances PSO had substantial outperformance rather than GA.

4.4. Impact of moves

In this section, we assess the impact of developed moves on the performance of the VNS algorithm. For this purpose, we remove a special move from the algorithm and perform it again to obtain new solutions in the given time limits. Since the best-found solutions for most instances were obtained within 60 s, in this section we consider $TL = 60$ seconds. The impacts of $\lambda - opt$, $\lambda - reverse$, $\gamma - swap$ and $\gamma - insert$ on the APD are summarized in Table 6. The APD of the VNS algorithm for all instances with $TL = 60$ seconds is 14.47 while the similar values for the algorithms VNS\{λ-opt}, VNS\{λ-reverse}, VNS\{γ-swap} and VNS\{λ-insert} are 23.5, 19.6, 19.5 and 17.3, respectively. As a result, it seems that the λ-opt move has the most improving impact on the performance of the VNS algorithm.

5. Conclusions and future research directions

In this paper, we considered a scheduling problem for a set of loading and unloading jobs with several types of products at a transshipment terminal, where the jobs arrive and leave the terminal in a set of batches. Furthermore, an inventory capacity for each type of product is considered. We developed a linear integer programming model and proposed an efficient VNS algorithm. For instances with $n \leq 30$, the VNS algorithm was able to find optimal solutions for 95% of test instances in less than one minute, and also it was capable of finding all within 180 s where the average and standard deviation of runtimes were 53 and 37 s, respectively. Moreover, we modified two previously developed metaheuristic algorithms, i.e. a GA and a PSO algorithm, to solve our problem. Comparative computational experiments and statistical analyses indicate that our developed VNS algorithm outperforms others.

For future research, we suggest considering the problem at hand with multi loading/unloading stations. In addition to this, developing exact solution approaches or other metaheuristic algorithms are interesting research topics.

CRediT authorship contribution statement

Mohammad Ranjbar: Supervision, Problem definition, Problem modelling, Developing algorithm, Computational analyses, Statistical Analyses, Writing - review & editing. **Reza Ghorbani Saber:** Model implementation, Developing algorithm, Implementation of algorithms, Computational results, Computational analyses, Statistical Analyses, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Table 6
Impact of the moves on the VNS algorithm within $TL = 60$ s.

n, pt	*Algorithm*				
	VNS	VNS\{λ-*opt*}	VNS\{λ-*reverse*}	VNS\{γ-*swap*}	VNS\{γ-*insert*}
$n = 10, pt = 2$	0.0	0.1	0.0	0.0	0.0
$n = 10, pt = 3$	0.0	0.3	0.0	0.1	0.0
$n = 10, pt = 4$	0.0	0.8	0.2	0.2	0.1
$n = 20, pt = 2$	1.5	3.5	2.6	2.5	2.1
$n = 20, pt = 3$	1.7	3.3	2.6	2.6	2.2
$n = 20, pt = 4$	1.8	3.5	3.1	2.9	2.5
$n = 30, pt = 2$	6.5	12.4	8.8	8.7	7.3
$n = 30, pt = 3$	7.5	12.7	9.1	9.2	8.5
$n = 30, pt = 4$	9.2	15.9	13.8	13.0	11.4
$n = 40, pt = 2$	27.3	41.2	35.4	36.7	31.2
$n = 40, pt = 3$	28.2	42.8	37.9	38.1	33.0
$n = 40, pt = 4$	31.8	50.0	40.8	39.5	36.7
$n = 50, pt = 2$	32.5	51.9	42.1	41.9	38.9
$n = 50, pt = 3$	33.7	55.3	45.8	46.7	41.1
$n = 50, pt = 4$	35.4	58.6	51.8	50.5	44.6

Acknowledgment

This work is supported by Ferdowsi University of Mashhad as a research project with number 45149 and date 22-10-2017.

References

[1] D. Briskorn, J.Y. Leung, Minimizing maximum lateness of jobs in inventory constrained scheduling, J. Oper. Res. Soc. 64 (2013) 1851–1864.
[2] R. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approxmimation in deterministic sequencing and scheduling: A Survey, Ann. Discrete Math. 5 (1979) 287–326.
[3] d. Briskorn, B.C. Choi, K. Lee, J. Leung, M. Pinedo, Complexity of single machine scheduling subject to nonnegative inventory constraints, Eur. J. Oper. Res. 207 (2010) 605–619.
[4] A. Bazgosha, M. Ranjbar, N. Jamili, Scheduling of loading and unloading operations in a multi stations transshipment terminal with release date and inventory constraints, Comput. Ind. Eng. 106 (2017) 20–31.
[5] C. Schwindt, N. Trautmann, Batch scheduling in process industries: an application of resource–constrained project scheduling, OR Spektrum 22 (2000) 501–524.
[6] K. Neumann, C. Schwindt, Project scheduling with inventory constraints, Math. Methods Oper. Res. 56 (2003) 513–533.
[7] K. Neumann, C. Schwindt, N. Trautmann, Scheduling of continuous and discontinuous material flows with intermediate storage restrictions, Eur. J. Oper. Res. 165 (2005) 495–509.
[8] J.H. Bartels, J. Zimmermann, Scheduling tests in automotive R & D projects, Eur. J. Oper. Res. 193 (2009) 805–819.
[9] P. Buijs, I.F. Vis, H.J. Carlo, Synchronization in cross-docking networks: A research classification and framework, Eur. J. Oper. Res. 239 (2014) 593–608.
[10] N. Boysen, M. Fliedner, Cross dock scheduling: Classification, literature review and research agenda, Omega 38 (2010) 413–422.
[11] N. Boysen, M. Fliedner, A. Scholl, Scheduling inbound and outbound trucks at cross docking terminals, OR Spectrum 32 (2010) 135–161.
[12] A.B. Arabani, M. Zandieh, S.M.F. Ghomi, Multi-objective genetic-based algorithms for a cross-docking scheduling problem, Appl. Soft Comput. 11 (2011) 4954–4970.
[13] N. Boysen, Truck scheduling at zero-inventory cross docking terminals, Comput. Oper. Res. 37 (2010) 32–41.
[14] W. Yu, P.J. Egbelu, Scheduling of inbound and outbound trucks in cross docking systems with temporary storage, Eur. J. Oper. Res. 184 (2008) 377–396.
[15] S. Forouharfard, M. Zandieh, An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems, Int. J. Adv. Manuf. Technol. 51 (2010) 1179–1193.
[16] B. Vahdani, R. Soltani, M. Zandieh, Scheduling the truck holdover recurrent dock cross-dock problem using robust meta-heuristics, Int. J. Adv. Manuf. Technol. 46 (2010) 769–783.
[17] N. Mladenović, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (1997) 1097–1100.
[18] R. Todosijević, A. Mjirda, M. Mladenović, S. Hanafi, B. Gendron, A general variable neighborhood search variants for the travelling salesman problem with draft limits, Optim. Lett. 11 (2017) 1047–1056.
[19] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, University of Michingan Press, Ann Arbor, MI, 1975.
[20] A. Boloori Arabani, M. Zandieh, S.M.T. Fatemi Ghomi, Multi-objective genetic-based algorithms for a cross-docking scheduling problem, Appl. Soft Comput. 11 (8) (2011) 4954–4970.
[21] D. Molavi, A. Shahmardan, M.S. Sajadieh, Truck scheduling in a cross docking systems with fixed due dates and shipment sorting, Comput. Ind. Eng. 117 (2018) 29–40.
[22] M. Tamannaeib, M. Rasti-Barzoki, Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem, Comput. Ind. Eng. 127 (2019) 643–656.
[23] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Vol. 1, 1995, pp. 39–43.
[24] A. Mohtashami, M. Tavana, F.J. Santos-Arteaga, A. Fallahian-Najafabadi, A novel multi-objective meta-heuristic model for solving cross-docking scheduling problems, Appl. Soft Comput. 31 (2015) 30–47.
[25] W. Wisittipanich, P. Hengmeechai, Truck scheduling in multi-door cross-docking terminal by modified particle swarm optimization, Comput. Ind. Eng. 113 (2017) 793–802.