Computers & Operations Research 138 (2022) 105585

Contents lists available at ScienceDirect

Computers and Operations Research

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

A new dynamic shape adjustment and placement algorithm for 3D yard allocation problem with time dimension

Tiantian Wang [a], Hong Ma [b,*], Zhou Xu [c], Jun Xia [d]

[a] School of Management, Zhejiang University, China
[b] School of Engineers, Zhejiang University, China
[c] Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hong Kong
[d] Sino-US Global Logistics Institute, Shanghai Jiao Tong University, China

ARTICLE INFO

Keywords:
Container terminals
Yard allocation
Time dimension
Dynamic programming
Heuristic

ABSTRACT

This paper studies a yard allocation problem at a container terminal, namely the 3D yard allocation problem with time dimension (3DYAPT), that determines the container storage locations in a given storage block to satisfy requirements from batches of arrived containers. The objective is to minimize the two-dimensional area of the storage block occupied for temporarily storing the containers within a given planning horizon (time dimension). The 3DYAPT is challenging and proved to be strongly NP-hard since it requires dynamically adjusting the shape of the allocated area when placing containers from the same request. We formulate the 3DYAPT as an integer linear programming model and develop a simulated annealing-based dynamic shape adjustment and placement algorithm (SA-DSAP). The simulated annealing-based algorithm comprises a novel dynamic programming procedure with several speed-up techniques that sequentially computes the storage space solution given a particular sequence of requests. Extensive computational experiments are conducted, showing that SA-DSAP is capable of finding optimal solutions very efficiently for nearly all small instances. For large instances, we also find that SA-DSAP produces significantly better heuristic solutions than the existing algorithm from the literature.

1. Introduction

Container terminals are large and critical components of complex logistic networks and are considered as strategic assets. Usually, export containers arrive at a container terminal through several modes of transportation, such as trucks and barges. Containers are dropped off at these terminals and are stored for a brief period of time. After this period, the containers are taken from the stack by cranes and loaded onto ships for delivery. This leads to the well-known yard allocation problem (YAP) faced by terminal operators (Fu et al., 2007; Zhen et al., 2016; Jin et al., 2016). Over the past few decades, technology has facilitated automation of container handling at terminals with tactical-level coordination in respect of allocation of terminal yard blocks, assignment of quay crane jobs and routes for the AGVs (automated guided vehicles) that travel between the quay sides and the yard blocks. Meanwhile, new decision support systems have also helped lower-level operations that determine whether and how containers should be stored within a specific yard block with limited space.

This paper focuses on yard space allocation for export containers. We study the 3D yard allocation problem with time dimension (3DYAPT), which was first introduced in Fu et al. (2007). The 3DYAPT is defined for a yard block, simply referred to as a yard in the rest of this paper, served by a few quay cranes and is assumed to be empty at the start of the planning horizon. A request is defined as the requirement from several batches of containers to be transported to the same destination. Each request has a storage time requirement and needs a certain set of yard space. Containers of the same request are usually stored adjacent to each other, and therefore containers in each request occupy a particular rectangular storage space. This rectangular storage space is not predetermined and is decided on arrival of the containers of different batches at different arrival times. From a time perspective, the area of this rectangular storage space should be non-decreasing, i.e., during the arrival of container batches, the area of the rectangular storage space either increases or remains unchanged as time progresses until all containers in the particular request are loaded for delivery to the same destination. As an example, Fig. 1(a) shows a storage solution to one valid request that spans from time point 2 to 6. It can be seen that the rectangular storage space increases from time point 2 to 5 because more containers to be taken to the

* Corresponding author.
E-mail address: hongma@zju.edu.cn (H. Ma).

https://doi.org/10.1016/j.cor.2021.105585
Received 31 August 2020; Received in revised form 18 August 2021; Accepted 26 September 2021
Available online 15 October 2021
0305-0548/© 2021 Elsevier Ltd. All rights reserved.

same destination arrive during this period. From time point 5 to 6, the storage space remains unchanged. This is possible because newly arrived containers can utilize the unused space in the storage solution up to time point 5. It is also easy to see that the rectangular storage spaces of different requests should be non-overlapping. For example, Fig. 1(b) shows a feasible storage solution to three requests where there is no storage space overlap. To optimize space utilization, the objective of the 3DYAPT is to minimize the largest storage space ever used throughout the planning horizon.

In extant literature on scheduling yard space utilization at container terminals, a major part of research has focused on tactical optimization, where yard sections (Lee et al., 2012; Lee and Jin, 2013), yard blocks (Zhang et al., 2003; Moccia et al., 2009), yard subblocks (Zhen et al., 2011; Tan et al., 2017; Jiang and Jin, 2017; Liu, 2020), yard bays (Lee et al., 2011; Ting and Wu, 2017) are considered and scheduled at a higher-level. Other studies concerning optimization of lower-level container handling operations mainly consider container weight (Kim et al., 2000; Kang et al., 2006), container type (Bazzazi et al., 2009; Gharehgozli and Zaerpour, 2018), container reshuffling with dynamic boundaries (Zhou et al., 2020) and yard facility deployment (Tao and Lee, 2015; Zhou et al., 2020), but to the best of our knowledge, very few of these works have looked into scheduling batches of containers with different arrival times.

It is worth mentioning that the 3DYAPT addressed in this paper has some similarities with the well-known bin packing problem though there are obvious differences. The three-dimensional bin packing problem (3DBPP) (Martello et al., 2000; Silva et al., 2019) can be regarded as a special case of the 3DYAPT, where the shape of the area for satisfying the space requirements is given, and the requirement of non-decrease of space over time is relaxed. While the 3DYAPT deals with containers allocation based on adjustable storage space shapes (items), in a general 3D packing, to the best of our knowledge, the shapes of items are not adjustable. On the other hand, both the 3DBPP and 3DYAPT have the objective of finding a compact layout of items, the ideas for solving the 3DBPP can be used in solving the 3DYAPT, such as the extreme point method (Wu et al., 2010) and the skyline method (Wei et al., 2011). However, because 3DYAPT possesses a time dimension that makes the item shapes irregular and adjustable, this special property then makes it impossible to apply methods used in 3DBPP directly to 3DYAPT. Therefore, it is necessary for us to design a new solution approach to cope with this special property of the 3DYAPT.

Chen et al. (2002) studied a yard allocation problem with time constraints (2DYAPT) that minimizes the utilized yard length. Each request in the 2DYAPT contains two dimensions, length and time. Similar to the 3DYAPT, 2DYAPT also requires the yard length not to decrease with time and there should be no overlap in yard length of different requests. Several metaheuristics, such as simulated annealing (SA), tabu search (TS), squeaky wheel optimization (SWO) and genetic algorithm (GA), are provided to obtain a near-optimal solution. Lim and Xu (2006) improved the solutions of the 2DYAPT by designing an effective critical shaking neighborhood search (CSNS) algorithm that picks and changes the order of a critical request in the sequence. Fu et al. (2007) extended the 2DYAPT to the 3DYAPT which considers both the length and the width of the yard as well as a time dimension. They argued that considering square-shaped storage spaces for containers is effective and a bottom-left strategy is adopted to allocate spaces to requests of containers. However, storage spaces generated by the bottom-left strategy are not always compact as they often lead to sparse unused space that can hardly be used for other requests. The square-shaped storage space is also an intuitive and simplified assumption that has an obvious limitation on space utilization in practice.

In this work, we consider a more general rectangular storage space which is different from extant literature on the 3DYAPT (Fu et al., 2007). We relax the square shape restriction on storage spaces, allowing rectangular shapes with adjustable lengths and widths. The idea is that rectangular storage shapes with adjustable lengths and widths offer much larger flexibility when fitting them into narrow spaces, compared to squares, thus having greater potential to generate more efficient yard allocation solutions. We prove that with the square shape restriction relaxed, the problem is still NP-Complete. We present a new integer linear programming formulation, and then propose an effective approach based on the simulated annealing algorithm and a novel dynamic shape adjustment placement procedure (SA-DSAP) that packs rectangular storage spaces into the yard to minimize the largest occupied area over the planning horizon. Our SA-DSAP consists of two stages. In the first stage, sequences of all requests are generated by SA. In the second stage, given a particular sequence, the storage space solution is computed using a dynamic programming (DP) procedure. Because, the DP procedure is to be frequently invoked, to further improve computational efficiency, we also develop some speed-up techniques eliminating storage spaces that are unlikely to be included in an optimal solution before they are evaluated by DP. Extensive computational experiments show that SA-DSAP is able to obtain optimal solutions for small-scale instances, and the solutions for large-scale instances are of much higher quality than the Bottom-Left heuristic used in Fu et al. (2007).

The remainder of this paper is organized as follows. In Section 2, we present the problem in greater detail and formulate the problem as an integer linear programming model. In Section 3, the general framework of the SA-DSAP is described, followed by details of its most critical component, the DSAP, in Section 4. Experiments and computational results are discussed in Section 5, and conclusions are drawn in Section 6.

2. Problem description and formulation for the 3DYAPT

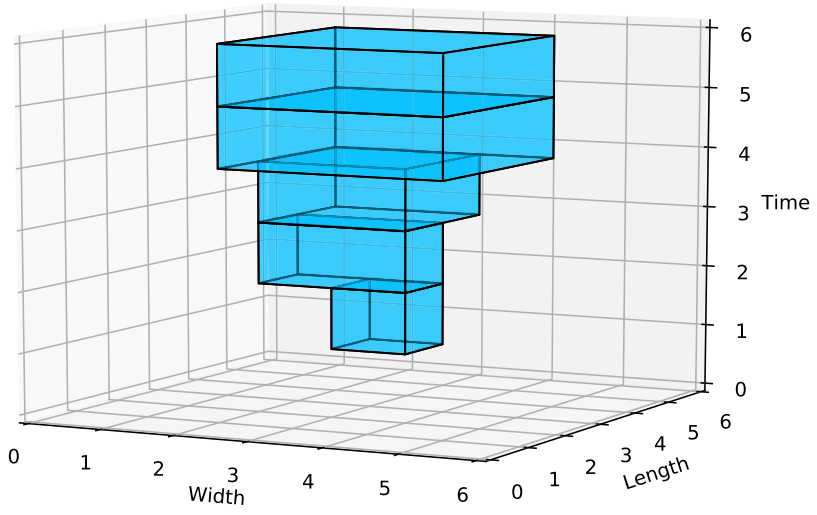
2.1. Problem description

To describe the 3DYAPT, let us consider a rectangular terminal yard that consists of $W \times L$ rectangular slots, where W and L represent the number of slots contained by the width and length of the yard, respectively. In each slot, it is possible to stack at most H standard containers. For ease of description, the rectangular yard area can be represented by the area $(0, 0, W, L)$, with coordinates $(0, 0)$ indicating the left-bottom corner point and the coordinates (W, L) indicating the right-top corner point.

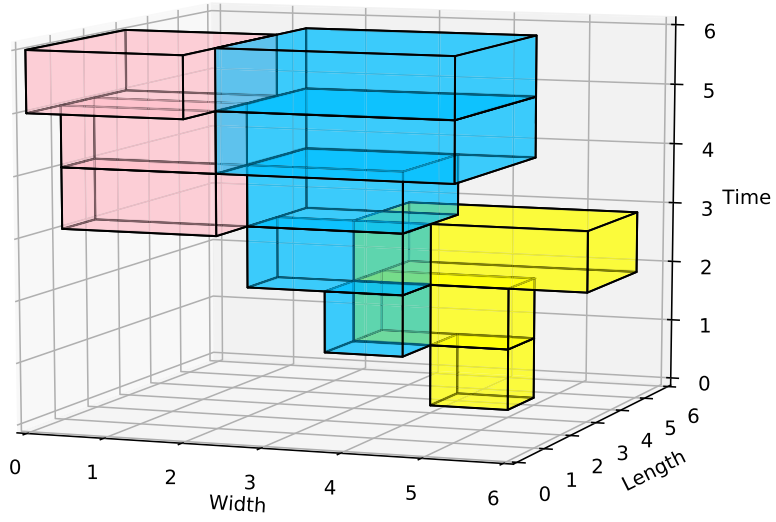
The considered time horizon T is composed of a series of evenly distributed discrete time points $T = \{1, 2, \ldots, |T|\}$. The arrival and departure of containers occur at a given time point. In case there are both arrival and departure during the same time unit, we assume that departure containers are handled first and then the arrival containers.

2.1.1. Request

We are given a set of Q containers whose arrivals and departures happen within time points in T. Containers for the same destination are collected in a group which we call as a *request*. Note that each container is included in exactly one request, and containers belonging to a request leave the yard area at the same departure time. Let $R = \{1, 2, \ldots, N\}$ $(N = |R|)$ denote the set of requests, where request $r \in R$ consists of n_r containers $(\sum_{r \in R} n_r = Q)$, and the departure time of each container involved is denoted by $t_r \in T$. We assume that the loading process of containers in a request can be completed in a short period, and the space occupied by request r can be released fully only after its departure time t_r. The arrival time of request r (denoted by a_r) is defined as the earliest arrival time among all arriving containers in request. Thus, we can define $T_r = \{a_r, a_r + 1, \ldots, t_r\}$ a subset of T, to represent the discretized time points occupied by request r. $n_{r,t}$ is defined as the total number of arrived containers in request r by time t. The storage space required by request r at time point t can be determined by $n_{r,t}$. t_r is a known input data which can be obtained from the output by Iris et al. (2018). Because containers may arrive at any time point $t \in T_r$ and are



(a) A valid request



(b) Three valid requests

Fig. 1. Examples of the three-dimensional requests.

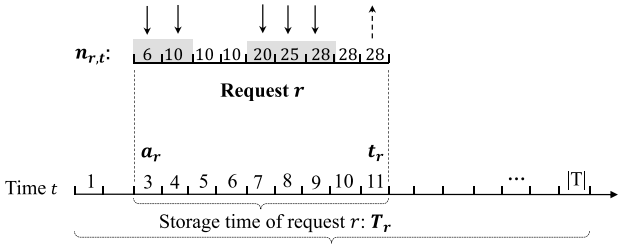


Fig. 2. Illustration of a request.

stored in the yard up to the delivery time of the request, $n_{r,t}$ must be non-decreasing, i.e., $n_{r,a_r} \leq n_{r,a_r+1} \leq \cdots \leq n_{r,t_r} = n_r$.

Fig. 2 shows an example of handling of a given request r within planning horizon T. Containers in this request r are stored in the yard from time point 3 ($a_r = 3$) to time point 11 ($t_r = 11$). In the beginning, 6 containers arrive at time point 3 followed by 4 more containers at time point 4. No container arrives between time points 5 and 6. At time points 7, 8 and 9, 10, 5 and 3 containers arrive at the yard respectively. The accumulated number of arrived containers $n_{r,t}$ from time point 3 to 11 is depicted in Fig. 2. We can see that, request r has a total of 28 arrived containers ($n_r = 28$) and all of them are released together at time point 11.

2.1.2. Feasible storage space

Given a request $r \in R$, we need to determine a rectangular space within the yard area to store the containers of request r, covering all time points in T_r. We denote $s_{r,t} = (x^-_{r,t}, y^-_{r,t}, x^+_{r,t}, y^+_{r,t})$ as the rectangular storage space at time $t \in T_r$, where $(x^-_{r,t}, y^-_{r,t})$ and $(x^+_{r,t}, y^+_{r,t})$ are coordinates of the left-bottom and the right-top corner points, respectively. At time t, the storage space $s_{r,t}$ is feasible when the following four conditions are satisfied.

(1) The rectangular storage space $s_{r,t}$ is fully contained in yard area, i.e., $0 \leq x^-_{r,t} < x^+_{r,t} \leq W$ and $0 \leq y^-_{r,t} < y^+_{r,t} \leq L$ must be satisfied simultaneously;

(2) The area occupied by storage space $s_{r,t}$ has enough slots for request r accumulated at time point t. This requirement can be expressed

by $n_{r,t} \le H(x^+_{r,t} - x^-_{r,t})(y^+_{r,t} - y^-_{r,t})$, where H is the maximum number of containers that can be stacked for each unit slot.

(3) Due to restraints imposed on yard cranes, the width $(x^+_{r,t} - x^-_{r,t})$ and the length $(y^+_{r,t} - y^-_{r,t})$ of a storage space need to be bounded by prescribed ranges $[\underline{w}, \overline{w}]$ and $[\underline{l}, \overline{l}]$, i.e., $\underline{w} \le x^+_{r,t} - x^-_{r,t} \le \overline{w}$ and $\underline{l} \le y^+_{r,t} - y^-_{r,t} \le \overline{l}$.

(4) To restrict the shape of the rectangular storage space, the difference between length $l_{r,t}(= y^+_{r,t} - y^-_{r,t})$ and width $w_{r,t}(= x^+_{r,t} - x^-_{r,t})$ of a feasible storage space $s_{r,t}$ should be within a predetermined range d, i.e., $|l_{r,t} - w_{r,t}| \le d$. It then should be noted that the square storage space case presented in Fu et al. (2007) corresponds to the case when $d = 0$, i.e., $|l_{r,t} - w_{r,t}| \le 0$, or $l_{r,t} = w_{r,t}$.

As shown in Fig. 1(a), there is a request r from time point 2 to 6, and at each time point, there is a corresponding three-dimensional feasible storage space. The yard area is assumed to be (0,0,6,6) and the height limit is 6 units, i.e., $L = 6$, $W = 6$ and $H = 6$. At time point 3, we assume there are 11 arriving containers ($n_{r,3} = 11$). Then, a feasible storage space at time point 3 is drawn as (2,2,3,3), ranging from the left-bottom point (2,2) to the right-top point (3,3). This storage space is fully contained in the yard and is large enough to accommodate 12 ($l \times w \times h$=(4-2)×(3-2)×6) containers at a time. Finally, the assumed length and width restrictions, $[\underline{w}, \overline{w}] = [1, 6]$ and $[\underline{l}, \overline{l}] = [1, 6]$, are also satisfied since the storage space has a length of 2 units and a width of 1 unit.

Because there exist different storage arrangements for the $n_{r,t}$ containers at each time t, we use $S_{r,t}$ to denote the set of all feasible storage spaces for request r at time t. Using an approach similar to Buhrkal et al. (2011), Iris et al. (2015), we simply generate $S_{r,t}$ for all requests at different time points a priori.

2.1.3. Constraints and objective

Let $\mathbf{s} = \{\mathbf{s}_1, \dots, \mathbf{s}_N\}$ denote a storage solution vector to all requests, where $\mathbf{s}_r = \{s_{r,a_r}, \dots, s_{r,t_r}\}$ $(s_{r,t} \in S_{r,t}, r \in R, t \in T_r)$ is the storage solution to request r. We aim to obtain a feasible storage solution vector $\mathbf{s}$ by allocating exactly one feasible storage space $s_{r,t}$ for each request r and time point t, such that the following constraints are satisfied:

(1) *Non-decreasing constraints*. As discussed earlier, for each request r, the rectangular spaces developed from time a_r to t_r must expand continuously to cover the arriving containers. Thus, given request $r \in R$ and given each two consecutive time points $t, t+1 \in T_r$, the two feasible storage spaces, $s_{r,t}$ and $s_{r,t+1}$, must satisfy $x^-_{r,t} \ge x^-_{r,t+1}$, $x^+_{r,t} \le x^+_{r,t+1}$, $y^-_{r,t} \ge y^-_{r,t+1}$, and $y^+_{r,t} \le y^+_{r,t+1}$, simultaneously;

(2) *Non-overlapping constraints*. These constraints stipulate that the feasible storage spaces of two different requests cannot share any area in the yard at any time points in T. Mathematically, given time $t \in T$, two requests r and r' such that $t \in T_r \cap T_{r'}$, the feasible storage spaces, $s_{r,t}$ and $s_{r',t}$, must satisfy at least one of the following: (i) request r' is completely on the left side of request r, i.e., $x^-_{r',t} \ge x^+_{r,t}$; (ii) request r' is completely on the right side of request r, i.e., $x^+_{r',t} \le x^-_{r,t}$; (iii) request r' is completely above request r, i.e., $y^-_{r',t} \ge y^+_{r,t}$; and (iv) request r' is completely under request r, i.e., $y^+_{r',t} \le y^-_{r,t}$.

In Fig. 1(b), the storage solution to the three requests is feasible. For each request, its feasible storage spaces increase or keep unchanged, which is in accord with the non-decreasing constraints. The feasible storage spaces do not overlap at each time point.

The 3DYAPT is then described as follows. Given the incoming containers grouped by different requests in R, the arrival and departure time points $a_r, t_r \in T_r$ for each request $r \in R$, and the sets $S_{r,t}$ $(r \in R, t \in T_r)$ of feasible storage spaces, the problem aims to select a series of feasible storage spaces $s_{r,t}$ $(r \in R, t \in T_r)$, such that the determined storage solution vector $\mathbf{s} = \{(s_{1,a_1}, \dots, s_{1,t_1}), \dots, (s_{N,a_N}, \dots, s_{N,t_N})\}$ satisfies both the non-decreasing constraints and the non-overlapping constraints. Our objective, which follows the definition of objective in Fu et al. (2007), is to minimize the area of the minimum rectangular cover s_C so as to maximize the utility of yard space. This minimum rectangular cover s_C can be represented by $(x^-_C, y^-_C, x^+_C, y^+_C)$, where $x^-_C = \min_{r \in R, t \in T_r} x^-_{r,t}$, $y^-_C = \min_{r \in R, t \in T_r} y^-_{r,t}$, $x^+_C = \max_{r \in R, t \in T_r} x^+_{r,t}$, and $y^+_C = \max_{r \in R, t \in T_r} y^+_{r,t}$.

Fig. 3(a) illustrates a feasible storage solution for the three requests. Fig. 3(b) shows the rectangular cover associated with the storage solution with an area of 48 (with = 6, length = 8). It can be observed that this rectangular cover is not optimal. It should also be noted that in 3DYAPT, containers are not required to be piled along the block side because it is hard to maintain such a requirement when requests keep arriving and getting released during the whole planning horizon. In Fig. 3(d), the optimal storage solution to the three requests is shown. The area of the optimal rectangular cover (Fig. 3(e)) is reduced to 18 (width = 3, length = 6).

2.2. Formulation

We now formulate the 3DYAPT as an integer linear programming model as follows.

2.2.1. Notations

Indices:

r: request, $r \in R$;
t: time point, $t \in T$;
p: position in the yard, $p \in P$;
s: rectangular storage space, $s := s_{r,t} \in S_{r,t}$. In the following passage, we use s to represent a storage solution $s_{r,t}$ as aforementioned.

Sets:

R: set of requests, $R = \{1, 2, \dots, N\}$;
T: set of time points of the planning horizon, $T = \{1, 2, \dots, |T|\}$;
T_r: set of time points of request r, $T_r = \{a_r, a_r + 1 \dots, t_r\}$, $r \in R$;
P: set of all positions in the yard, $P = \{(x_p, y_p) : x_p \in \{1, 2, \dots, W\}, y_p \in \{1, 2, \dots, L\}\}$;
$S_{r,t}$: set of candidate storage solutions for request r at time point t, $S_{r,t} = \{s_{r,t}\} = \{s\}$, $r \in R$, $t \in T_r$;
R_t: set of requests that requires storage space at time point t, $R_t = \{r | r \in R, t \in T_r\}$, $R_t \subset R$, $t \in T$.

Parameters:

x_p: x-coordinate of position p;
y_p: y-coordinate of position p;
$\theta_{p,s}$: equals one if position p is in rectangular storage space s; otherwise, equals zero, $p \in P$, $s \in S_{r,t}$.

Decision variables:

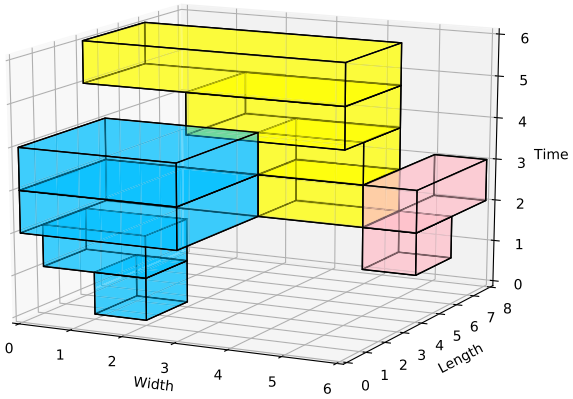
$\pi_p \in \{0, 1\}$: set to one if position p is the point at the right-top corner of rectangular cover s_C, and zero otherwise, $p \in P$. $s_C = (x^-_C, y^-_C, x^+_C, y^+_C)$ is the minimum rectangular cover as defined in Section 2.1.3.

$\lambda_{r,t,s} \in \{0, 1\}$: set to one if the rectangular storage space selected for request r at time t is s, and zero otherwise, $r \in R$, $t \in T_r$, $s \in S_{r,t}$.

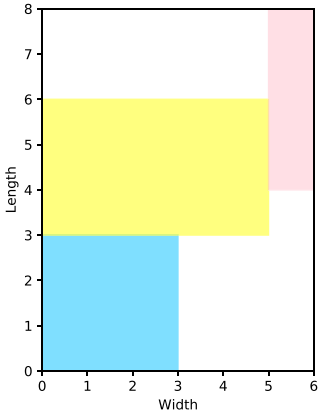
Here, P is the set of positions in the yard. Because the yard itself is a two-dimensional rectangular area from left-bottom point $(0, 0)$ to right-top point (W, L) (Fig. 4(a)), considering the unit container size, coordinates for all occupied rectangular areas can be defined as integers. Furthermore, we use the right-top corner point of a rectangular area to indicate its position in the yard (Fig. 4(b)). Thus, set P is denoted as $P = \{(x_p, y_p) : x_p \in \{1, 2, \dots, W\}, y_p \in \{1, 2, \dots, L\}\}$. It is obvious that the right-top corner point (x^+_s, y^+_s) of any feasible storage space s belongs to set P.

R_t is the set of requests that are stored in the yard at time point t, i.e., $R_t = \{r : r \in R, a_r \le t \le t_r\}$. Hence, R_t is a subset of the overall request set R, i.e., $R_t \subset R$. $\theta_{p,s}$ is an indicator which equals 1 when position p is covered by storage space s. Recall that $s = (x^-_s, y^-_s, x^+_s, y^+_s)$. The value of $\theta_{p,s}$ can be obtained from coordinates $p = (x_p, y_p)$ and coordinates of the left-bottom and right-top corner points (x^-_s, y^-_s), (x^+_s, y^+_s) of s as follows:

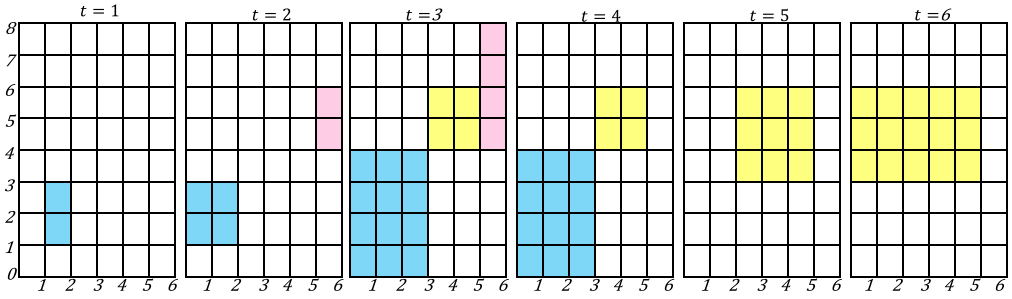
$$\theta_{p,s} = \begin{cases} 1 & \text{if } x^-_s < x_p \le x^+_s, y^-_s < y_p \le y^+_s, \\ 0 & \text{otherwise.} \end{cases}$$



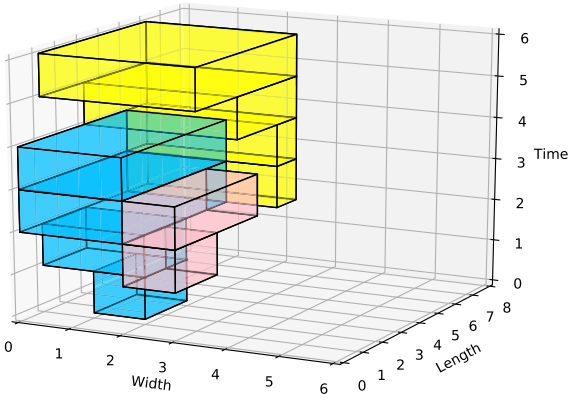
(a) Feasible solution



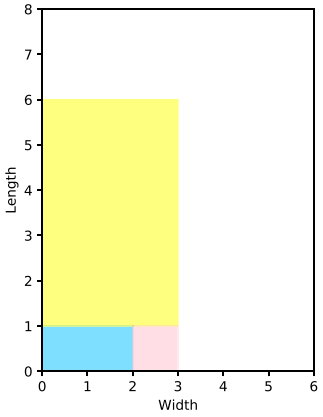
(b) Feasible area



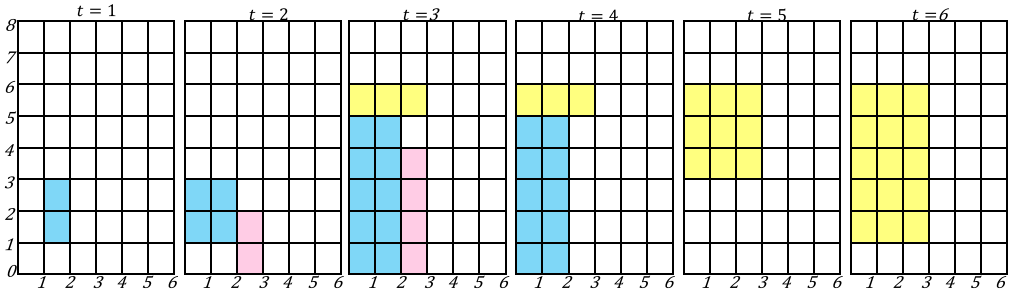
(c) 2D views of Figure 3a



(d) Optimal solution



(e) Optimal area



(f) 2D views of Figure 3d

Fig. 3. Illustration of storage space solutions to three requests.

2.2.2. Mathematical model

The objective of the 3DYAPT is to minimize the area of the rectangular cover s_C for all requests R in time horizon T. As defined, the rectangular cover is $s_C = (x_C^-, y_C^-, x_C^+, y_C^+)$, and the points at the left-bottom and the right-top corners of s_C are $p^- = (x_C^-, y_C^-)$ and $p = (x_C^+, y_C^+)$ respectively. Noting that the left-bottom point p^- is fixed to $(0, 0)$ by minimization, we then can formulate the objective with only

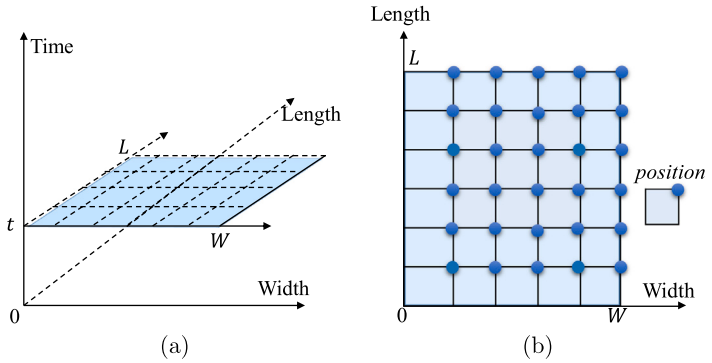


Fig. 4. Illustration of positions.

the right-top point p ($p \in P$) as follows:

$$\min \sum_{p\in P} x_p y_p \pi_p. \tag{1}$$

Objective (1) minimizes the yard space ever used by all requests during the whole planning horizon.

Each request has to satisfy the non-decreasing constraint, that is its rectangular storage space increases or keeps unchanged with time. Considering two consecutive time points t and $t+1$ of request r ($r \in R, t \in \{1,\dots,t_r-1\}, t+1 \in \{2,\dots,t_r\}$), we have two consecutive rectangular spaces s and s'. The equation $\sum_{s\in S_{r,t}} \lambda_{r,t,s} = 1$ ($r \in R, t \in T_r$) guarantees that one feasible storage space is exactly assigned to a request at a time point. To ensure that the storage space s assigned to request r at time t is occupied until departure time t_r, the storage space s' at time $t+1$ should be larger or at least equal to s. That is, the left edge of s' should be left to that of s, the right edge of s should be right to that of s, the bottom edge of s' should be lower than that of s, and the top edge of s' should be higher than that of s. So the non-decreasing constraints are formulated as follows:

$$\sum_{s\in S_{r,t}} \lambda_{r,t,s} = 1, \quad r \in R, t \in T_r, \tag{2}$$

$$\sum_{s'\in S_{r,t+1}} x^-_{s'} \lambda_{r,t+1,s'} \le \sum_{s\in S_{r,t}} x^-_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{3}$$

$$\sum_{s'\in S_{r,t+1}} y^-_{s'} \lambda_{r,t+1,s'} \le \sum_{s\in S_{r,t}} y^-_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{4}$$

$$\sum_{s'\in S_{r,t+1}} x^+_{s'} \lambda_{r,t+1,s'} \ge \sum_{s\in S_{r,t}} x^+_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{5}$$

$$\sum_{s'\in S_{r,t+1}} y^+_{s'} \lambda_{r,t+1,s'} \ge \sum_{s\in S_{r,t}} y^+_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{6}$$

Constraints (2) ensure that one storage space is assigned to a request at a time. Constraints (3)–(6) are the non-decreasing constraints ensuring that the storage space for each request either increases or remains unchanged during its storage time interval.

Constraints (7) are the non-overlapping constraints ensuring that at a given time, each storage space can be used by at most one request. According to the non-overlapping constraint, at time point t, any two different requests r and r' ($r, r' \in R_t$) cannot occupy the same space simultaneously. Thus, the sets of positions of their storage spaces $s_{r,t}$ and $s_{r',t}$ should be disjoint. That is, at time point t, for any position $p \in P$, it must satisfy either of the following two conditions: 1) p is covered by $s_{r,t}$ or $s_{r',t}$, but not both; 2) p is neither covered by $s_{r,t}$ nor $s_{r',t}$. Then the non-overlapping constraints are formulated as follows:

$$\sum_{r\in R_t} \sum_{s\in S_{r,t}:\theta_{p,s}=1} \lambda_{r,t,s} \le 1, \quad t \in T, p \in P, \tag{7}$$

where the left-side expression represents the total number of times that point p is covered by all rectangular storage spaces at time t that cover point p.

$$x^+_s \lambda_{r,t,s} \le \sum_{p\in P} x_p \pi_p, \quad r \in R, t \in T_r, s \in S_{r,t}, \tag{8}$$

$$y^+_s \lambda_{r,t,s} \le \sum_{p\in P} y_p \pi_p, \quad r \in R, t \in T_r, s \in S_{r,t}, \tag{9}$$

$$\sum_{p\in P} y_p \pi_p \le L, \tag{10}$$

$$\sum_{p\in P} x_p \pi_p \le W, \tag{11}$$

$$\sum_{p\in P} \pi_p = 1, \tag{12}$$

Constraints (8) and (9) ensure that the rectangular cover s_C is valid to cover the storage spaces of all requests during the whole planning horizon. Constraints (10)–(11) guarantee that all requests are stored within the yard area. Constraint (12) determines the size of the rectangular cover s_C. The left-bottom corner point of rectangular cover s_C is restricted to point $(0,0)$ by Constraints (8)–(9) and Objective (1).

$$\lambda_{r,t,s} \in \{0,1\}, \quad r \in R, t \in T_r, s \in S_{r,t}, \tag{13}$$

$$\pi_p \in \{0,1\}, \quad p \in P, \tag{14}$$

Constraints (13)–(14) define integral decision variables. As a result, we obtain a mathematical model **M1** with objective (1) and constraints (3)–(14) for the 3DYAPT.

As mentioned in Section 2.1.2, it is easy to see that the squared storage space case presented in Fu et al. (2007), which is strongly NP-hard, is a special case of the 3DYAPT. Therefore, the 3DYAPT is strongly NP-hard.

3. Solving the 3DYAPT by simulated annealing

In this section, we present a simulated annealing (SA)-based heuristic solution approach for the 3DYAPT. The main component of this solution approach is a dynamic programming algorithm based on the dynamic shape adjustment placement (DSAP) procedure. DSAP determines the optimal storage spaces for arriving containers from a set of feasible storage spaces.

According to the model presented in Section 2, the 3DYAPT consists of two decisions: (1) the storage shape-related decision, and (2) the storage position-related decision for each request at each storage time point. Both decisions are represented by the binary variables $\lambda_{r,t,s}$, ($r \in R, t \in T_r, s \in S_{r,t}$). Other decision variables π_p ($p \in P$) regarding the objective value are dependent on $\lambda_{r,t,s}$. In our heuristic solution approach, decision $\lambda_{r,t,s}$ is determined by deciding a feasible storage space $s_{r,t} \in S_{r,t}$ for each request r at time point t. We present each feasible solution to our 3DYAPT by a storage solution vector $\mathbf{s} = \{\mathbf{s}_1,\dots,\mathbf{s}_N\}$ where $\mathbf{s}_r = \{s_{r,a_r},\dots,s_{r,t_r}\}$ ($r \in R, s_{r,t} \in S_{r,t}, t \in T_r$) is composed of the decided feasible storage spaces as defined in Section 2.1.

3.1. Preprocessing: generating set $S_{r,t}$

We first introduce the preprocessing step which converts the accumulated number of arrived containers $n_{r,t}$ ($r \in R, t \in T_r$) into a set of feasible storage spaces $S_{r,t}$ for each request r and time point t. These feasible storage spaces in set $S_{r,t}$ are to be directly used in the solution approach. We define parameter $\Omega_{r,t}^{min}$ as the minimum yard area needed to store $n_{r,t}$ containers for request r at time t, which is computed as $\Omega_{r,t}^{min} = \lceil n_{r,t}/H \rceil$. Accordingly, the set of feasible storage spaces $S_{r,t}$ can be generated by an enumeration procedure, which enumerates all feasible storage spaces to satisfy the following three conditions: (1) each feasible storage space $s_{r,t}$ should not exceed the yard limits; (2) the area of a storage space $s_{r,t}$ should be no less than $\Omega_{r,t}^{min}$; and (3) each $s_{r,t} \in S_{r,t}$ should not be dominated by any other storage space in set $S_{r,t}$.

Algorithm 1: Framework of SA-DSAP.

1: **procedure** SA-DSAP($n_{r,t}, R, W, L$)
2: Preprocessing: convert $n_{r,t}$ to set $S_{r,t}$ ($r \in R, t \in T_r$);
3: Generate an initial sequence p_0 of the N requests;
4: $\mathbf{s}_0, \Omega_0 \leftarrow$ DSAP(p_0, R, W, L);
5: Let $p, p^* \leftarrow p_0$, $\Omega, \Omega^* \leftarrow \Omega_0$ and $\mathbf{s}, \mathbf{s}^* \leftarrow \mathbf{s}_0$;
6: Generate initial temperature $\tau_0 = \alpha \times \Omega$;
7: Let $\tau = \tau_0$;
8: **while** $\tau \geq 0.001$ **do**
9: Add τ to temperate schedule $\mathcal{T}$;
10: $\tau = \beta \times \tau$;
11: **for** each temperature τ in $\mathcal{T}$ **do**
12: Let $K = K_{max}, I = 0$;
13: **while** $K > 0$ **do**
14: Call a two-swap operator to generate p' from p;
15: $\mathbf{s}', \Omega' \leftarrow$ DSAP(p', R, W, L);
16: **if** $\Omega' < \Omega^*$ **then**
17: $I = 0$
18: **else**
19: $I = I + 1$
20: **if** $I \geq I_{max}$ **then**
21: break;
22: **if** $\Omega' < \Omega$ **then**
23: $p \leftarrow p'$, $\Omega \leftarrow \Omega'$ and $\mathbf{s} \leftarrow \mathbf{s}'$;
24: **if** $\Omega < \Omega^*$ **then**
25: $p^* \leftarrow p$, $\Omega^* \leftarrow \Omega$ and $\mathbf{s}^* \leftarrow \mathbf{s}$;
26: **else if** $random[0,1] < exp[(\Omega - \Omega')/\tau]$ **then**
27: $p \leftarrow p'$, $\Omega \leftarrow \Omega'$ and $\mathbf{s} \leftarrow \mathbf{s}'$;
28: **return** $\mathbf{s}^*, \Omega^*$

3.2. General framework of SA-DSAP

The simulated annealing based dynamic shape adjustment placement algorithm, namely SA-DSAP, is presented in Algorithm 1. It iteratively explores and adjusts sequences of requests within the SA framework. The sequence of requests is the sequence for allocating yard space. For each sequence of requests explored, it applies the dynamic shape adjustment placement algorithm (DSAP) to determine the feasible storage solutions $\mathbf{s}_r$ ($r \in R$) for every single request, one by one. For each explored sequence p of requests, a feasible storage solution vector $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$ and its relevant objective value a are computed by the DSAP algorithm. Details of the DSAP are described in Section 4. The results obtained by DSAP are utilized by the simulated annealing for further exploration of new sequences of requests.

We generate an initial sequence p_0 of requests by the following two rules: (1) Generate the sequence in decreasing order in terms of the number of containers since request with more containers usually takes more storage space; (2) For two or more requests with the same number of containers, the sequence is generated in increasing order of the size of set $S_{r,t}$. The intuition behind the rules is that requests with smaller sizes are considered as having less flexibility in determining the required storage space. Hence, they are placed in the front of the sequence p.

In SA-DSAP, we adopt a SA-based framework to iteratively adjust the sequence of requests. The initial temperature is set to the product of parameter α (=0.382) and the objective value of the initial sequence of requests. Each following temperature is set to the current temperature value multiplied by a decaying parameter β (=0.618). The annealing process stops when the temperature drops below 0.001. For each temperature τ, we apply a random two-swap operator to sequence p to get a neighbor p'. The storage solution vector of p' is accepted when its objective Ω' outperforms the objective Ω of current sequence p ($\Omega' < \Omega$). When p' is not better than p ($\Omega' \geq \Omega$), it is still to be accepted with a probability $exp[(\Omega - \Omega')/\tau]$. At each temperature, the number of iterations is limited to K_{max} (=500) times. To balance search efficiency and solution quality, this searching process is terminated if the objective value has no improvement after I_{max} (=400) consecutive iterations.

4. Detailed design of DSAP

4.1. Main idea

Given a sequence of requests $p = \{p_1, p_2, \ldots, p_N\}$, DSAP determines storage solutions $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ for requests one by one, where $\mathbf{s}_i$ is the storage solution to request r ($r = p_i$) and is composed of a series of storage space decisions at each of its handling time point t, i.e., $\mathbf{s}_i = \{s_{r,t} : r = p_i, a_r \leq t \leq t_r\}$.

As shown in Algorithm 2, DSAP solves the N requests sequentially by using N iterations following sequence p. Each iteration determines the storage solution for a request by minimizing the yard area used up to this request. In each iteration i, the algorithm takes the storage solutions $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}$ of its preceding requests as input, and then generates the storage solution $\mathbf{s}_i$ for request p_i. To make the algorithm consistent, we use a dummy solution $\mathbf{s}_0$ to represent the empty yard before request p_1 is assigned. After solving the ith iteration, the obtained storage solution $\mathbf{s}_i$ of request p_i is saved before proceeding to the next iteration, and the available yard spaces at each time point are updated (Line 12). At each time t, we use a $W \times L$ binary matrix to indicated two-dimensional yard status. Each element of the binary matrix represents a container space unit, which is set to 1 if the space is occupied at time t and set to 0 otherwise. In DSAP, once the storage spaces s_i for the ith request in sequence p is determined, the yard space status in then updated by setting all the space units of s_i to 1.

Algorithm 2: Generating storage solutions by DSAP given sequence p.

1: **procedure** DSAP(p, R, W, L)
2: $L_0 = 0, W_0 = 1$
3: $\mathbf{s}_0$: empty yard
4: **for** $i \leftarrow 1$ **to** N **do**
5: **for** $w \leftarrow W$ **to** W_{i-1} **do**
6: $L_i(w), \mathbf{s}_i(w) \leftarrow$ SolveRequest($i, p, R, w, L_{i-1}, \{\mathbf{s}_0, \ldots, \mathbf{s}_{i-1}\}$)
7: $\Omega_i(w) = L_i(w) \times w$
8: $\Omega_i = \min_{W_{i-1} \leq w \leq W} \Omega_i(w)$
9: $W_i = \arg \min_{W_{i-1} \leq w \leq W} \Omega_i(w)$
10: $L_i = L_i(W_i)$
11: $\mathbf{s}_i = \mathbf{s}_i(W_i)$
12: UpdateYardStatus
13: $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$, $\Omega = \Omega_N$
14: **return** $\mathbf{s}, \Omega$

In the ith iteration, storage solution $\mathbf{s}_i$ is deduced by selecting a feasible storage solution which leads to the minimum rectangular

yard area a_i used by requests $p_1, \ldots, p_i$. The area a_i is computed as $L_i \times W_i$, where L_i and W_i are the yard length and yard width ever reached by these requests. We use a three-step procedure to generate $\mathbf{s}_i$. First, the used width W_i is artificially fixed to a feasible value w in range $[W_{i-1}, W]$, where W_{i-1} is the yard width reached by requests $p_1, \ldots, p_{i-1}$ and W is the yard width boundary. The value w is reversely looped from W to W_{i-1} such that yard space is assigned according to width before according to length (Line 5). Second, given that W_i is fixed to value w, the yard length $L_i(w)$ ever reached by requests $p_1, \ldots, p_i$ is solved by the `SolveRequest` procedure based on dynamic programming. At the same time, the storage solution $\mathbf{s}_i(w)$ is obtained (Line 6). The area used by $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}, \mathbf{s}_i(w)$ is computed as $\Omega_i(w) = L_i(w) \times w$ (Line 7). Finally, a_i, W_i, L_i and storage solution $\mathbf{s}_i$ for request p_i are derived according to the w value that leads to the minimum area as W_i, i.e., $W_i = \arg\min_{W_{i-1} \le w \le W} \Omega_i(w)$ (Line 8~11). After N iterations, the storage solutions $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ and yard area used Ω ($\Omega = \Omega_N$) following sequence p can be obtained (Line 13).

4.2. Computing the minimum used length

As mentioned in line 6 of Algorithm 2, the `SolveRequest` procedure computes the yard length $L_i(w)$ ever reached by requests $p_1, \ldots, p_i$ when storage solutions of requests $p_1, \ldots, p_{i-1}$ are fixed to $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}$ and the yard width W_i reached by requests $p_1, \ldots, p_i$ is fixed to w. In this subsection, we present details of the dynamic programming (DP) procedure in the `SolveRequest`, illustrating how the storage solution $\mathbf{s}_i(w)$ for request r ($r = p_i$) is determined (Algorithm 3).

Algorithm 3: Generating solution $\mathbf{s}_i$ to the ith request.

1: **procedure** SolveRequest($i, p, R, w, L_{i-1}, \{\mathbf{s}_0, \ldots, \mathbf{s}_{i-1}\}$)
2: Get information of request r: $a_r, t_r, S_{r,t}$ ▷ $r = p[i]$
3: **for** stage $k = |T_r|$ **to** 1 **do**
4: **for** decision $s \in S_k$ **do**
5: Computing state value $h(k, s)$
6: $L_i(w) = \min_{\forall s \in S_1} h(1, s)$
7: $\mathbf{s}_i(w) = \{s_1^*, \ldots, s_{|T_r|}^*\} \leftarrow \arg\min_{s \in S_1} h(1, s)$
8: **return** $L_i(w), \mathbf{s}_i(w)$

Definition of stage. Since request r covers $|T_r|$ ($|T_r| = t_r - a_r + 1$) time points, we define $|T_r|+1$ stages in the dynamic programming, that is, from stage 0 to stage $|T_r|$. While stage 1 to stage $|T_r|$ correspond to time point a_r to t_r, stage 0 is a dummy stage representing a time point that is after the last request is scheduled and before the current request that is to be scheduled.

Definition of decision. At each stage, a storage space needs to be determined. The decision s at stage k, defined as a feasible storage space $s_{r,k+a_r-1}$ ($r = p_i$), determines the storage space assigned to the request at stage k. Accordingly, the set of decisions S_k at stage k is the set of all storage spaces $S_{r,t}$ at time t, i.e., $S_k = S_{r,k+a_r-1}$.

Definition of state. For request p_i, a state (k, s) is defined as taking decision s at stage k. We define the state value $h(k, s)$ as the minimum yard length to satisfy the space requirement of request p_i for time points $k + a_r - 1, \ldots, t_r$ for state (k, s).

In the DP, the state values are computed backward from stage $|T_r|$ to stage 1. At the beginning of stage k, all the states at stages $k+1, \ldots, |T_r|$ are known. In order to obtain state values $h(k, s)$ for each state at stage k, we consider the state transition from states $(k + 1, s)$ ($s \in S_{k+1}$) to state (k, s). So, the computation of the state value $h(k, s)$ is based on the following cases:

Case 1: State value $h(0, s)$ at the dummy stage 0. The decision set of dummy stage 0 is empty, and the state value at stage 0 is equal to the minimum used yard length after requests $p_1, \ldots, p_{i-1}$ are scheduled, that is $h(0, s) = L_{i-1}$. L_0 is simply set as zero for convenience.

Case 2: State value $h(k, s)$ of infeasible decisions at stage k ($k \ge 1$). In the preprocessing stage of the heuristic method, the storage space candidates set $S_{r,t}$ is generated without checking its feasibility. Hence, decisions in set S_k may violate the non-overlapping constraints or the non-decreasing constraints. To overcome this issue, a two-step preprocessing procedure is applied before they are used to compute $h(k, s)$.

- Step 1: We first consider the non-overlapping constraints. If the storage space corresponding to decision s overlaps with the scheduled yard space at that time point, $h(k, s)$ is then set to a very large positive integer M.
- Step 2: We next consider the non-decreasing constraints. First, the decisions at stage $|T_r|$ all satisfy the non-decreasing constraints. Next, for decision s at stage $|T_r| - 1$, if there exists a decision s' at stage $|T_r|$ such that the storage space of s' covers the storage space of s, decision s is then a feasible decision at stage $|T_r| - 1$. We define set $S_{|T_r|}(s)$, namely the cover set of decision s, as the set of all decisions at stage $|T_r|$ that cover decision s at stage $|T_r| - 1$. It is easy to see that $S_{|T_r|}(s)$ is a subset of the decision set $S_{|T_r|}$. Generally, for a decision s at stage k ($k = 1, \ldots, |T_r| - 1$), its cover set is $S_{k+1}(s)$ ($S_{k+1}(s) \subset S_{k+1}$). If $S_{k+1}(s)$ is empty, decision s is infeasible and the value $h(k, s)$ is set to a very large positive integer M.

In Case 3 and Case 4, computation of $h(k, s)$ only involves feasible decisions, which is also processed backward from stage $|T_r|$ to stage 1.

Case 3: State value $h(|T_r|, s)$ of feasible decisions at stage $|T_r|$. At stage $|T_r|$, the state value $h(|T_r|, s)$ of choosing decision s takes the greater value of $h(0, s)$ and $y_{r,t}^+$, that is $h(|T_r|, s) = \max\{L_{i-1}, y_{r,t}^+\}$. $h(0, s)$ is equal to L_{i-1} as defined in Case 1. $y_{r,t}^+$ is the yard length corresponding to the feasible storage space when decision $s = (x_{r,t}^-, y_{r,t}^-, x_{r,t}^+, y_{r,t}^+)$.

Case 4: State value $h(k, s)$ of feasible decisions at stage k ($k < |T_r|$). As defined in Case 2, a feasible decision s at stage k has a cover set $S_{k+1}(s)$ at stage $k + 1$. From decision s' ($s' \in S_{k+1}(s)$) at stage $k + 1$ to decision s at stage k, the used yard length by s is equal to $h(k + 1, s)$ due to the non-decreasing constraints. $h(k, s)$ takes the minimum value among all transitions from s' to s. Therefore, from all feasible decisions at stage $k + 1$ to decision s at stage k, it follows that $h(k, s) = \min_{s' \in S_{k+1}(s)} h(k + 1, s')$, and the selected decision s_{k+1}^* at stage $k + 1$ equals $arg\min_{s' \in S_{k+1}(s)} h(k + 1, s')$.

Based on the above four cases, noting that the feasible storage space s is defined as $(x_{r,t}^-, y_{r,t}^-, x_{r,t}^+, y_{r,t}^+)$, the recurrent equation to determine the value of $h(k, s)$ is as follows:

$$h(k,s) = \begin{cases} L_{i-1}, & \text{if } k = 0, \\ M, & \text{if overlap or decreasing,} \\ \max\{L_{i-1}, y_{r,t}^+\}, & \text{if } k = |T_r|, \\ \min_{s' \in S_{k+1}(s)} h(k+1, s'), & \text{if } k = 1, \ldots, |T_r| - 1, \end{cases}$$

After obtaining all state values of $h(1, s)$ ($s \in S_1$) at stage 1, we set the minimum of them as the final objective value $L_i(w)$ for a given yard width w (Line 6 in Algorithm 3). The storage solution $\mathbf{s}_i(w) = \{s_1^*, \ldots, s_{|T_r|}^*\}$ is then derived through backtracking of states from stage 1 to stage $|T_r|$ (Line 7).

4.3. Speed-up techniques

In the aforementioned DP, we compute $|S_k|$ states at each stage k ($1 \le k \le |T_r|$). Recall that set S_k equals the storage space set $S_{r,t}$ ($t = a_r + k - 1$). Thus, to get the minimum used yard length $L_i(w)$ when W_i is fixed to w, DP has to compute the state values for $\sum_{k=1}^{|T_r|} |S_k| = \sum_{t=t_r}^{a_r} |S_{r,t}|$ times. Set $S_{r,t}$ contains a huge number of feasible storage spaces for the current request at time point t following the preprocessing procedure discussed in Section 3. However, it is easy to see that some feasible storage spaces $s_{r,t}$ are not good enough to generate a small yard length, so the DP procedure can be accelerated

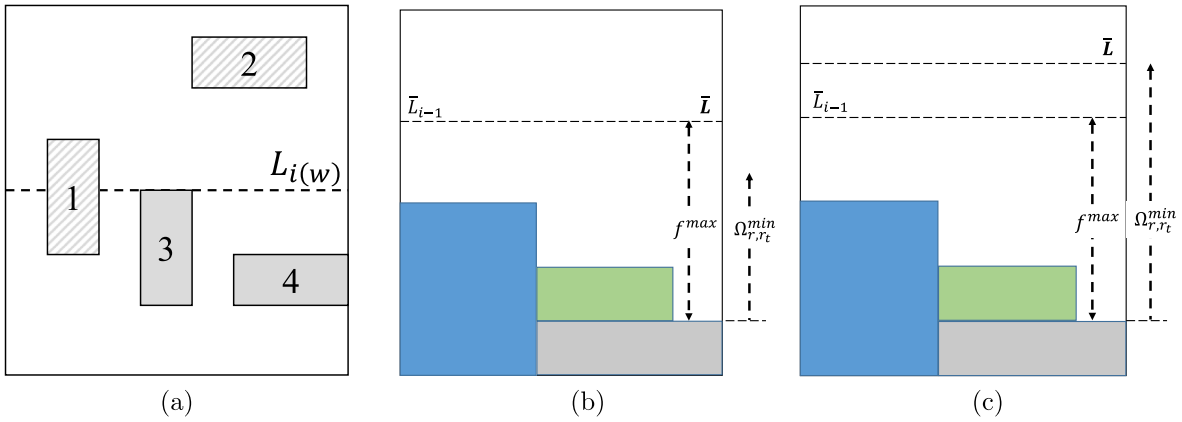


Fig. 5. Illustration of the speed-up process.

simply by ignoring such storage spaces. As an example, in Fig. 5, storage spaces 1 and 2 should be ignored due to their inefficiency while storage spaces 3 and 4 should be retained.

We then compute an estimated value $\bar{L}$ for $L_i(w)$ to help determine whether a feasible storage space should be evaluated or ignored. If this estimated value $\bar{L} < L_i(w)$, the DP procedure would be a waste of time searching the reduced set without finding any feasible solution. If $\bar{L}$ is much larger than $L_i(w)$, the reduced set would still possibly be too large. Therefore, we aim to set $\bar{L}$ to a value larger than $L_i(w)$ by an appropriate amount. The following three steps are then taken.

(1) Check the available yard spaces according to the minimum length $\bar{L}_{i-1}$ for the preceding requests at time point t_r. We only consider available yard spaces at t_r because the minimum yard length at any other time point t ($a_r \le t \le t_r - 1$) is bounded by that at time point t_r due to the non-decreasing constraints. Yard length $\bar{L}$ could be reached by a particular feasible storage space in set S_{r,t_r}.

(2) If the special feasible storage space s (width = 1, length = Ω^{min}_{r,t_r}) can be assigned to the maximum available yard space f^{max} ($\Omega^{min}_{r,t_r} \le f^{max}$), yard length would not change, i.e., $\bar{L} = \bar{L}_{i-1}$ as illustrated by Fig. 5(b). Otherwise, $\bar{L}$ is set to a minimum possible value such that s can be placed into the available yard space (Fig. 5(c)).

(3) Run the DP with the reduced storage spaces. If no feasible solution is found, increase $\bar{L}$ by 1, until a feasible solution is obtained.

5. Computational experiments and results

In this section, we first discuss test instance generation. Then, a simple lower bound (LB) to the objective value for each test instance is introduced. Next, we compare the performance of our solution approach with the simulated annealing based bottom-left (SA-BL) algorithm used in Fu et al. (2007) on both small, medium and large-scale instances. Finally, our solutions are compared with the optimal solutions obtained by a general optimization solver, IBM ILOG CPLEX, for small-scale instances. All experiments are implemented in Java and are conducted on a CentOS Linux workstation equipped with 16GB memory and Intel Pentium processor running at 3.5 GHz. The reported computational times of SA-DSAP include the processing time of generating set $S_{r,t}$. We use IBM ILOG CPLEX 12.61 for solving the ILP models.

5.1. Test data generation

Eight instances were developed and tested in Fu et al. (2007), but they are no longer available. Following the description in Fu et al. (2007) and taking new practical data (Port of Hong Kong in Figures 2019 Edition) into consideration, we generate 13 groups of new test instances, G1~G13, and each group consists of five test instances (available on GitHub[1]). These instances are all generated in a yard block with fixed width ($W = 6$), length ($L = 15$). The maximum stack height H is set to 5.

The planning horizon considered is set as seven days. Each day consists of 24 time points for one hour each. In total, there are 168 time points for the planning horizon. The number of requests ranges from 5 to 65 requests among Instance Groups G1 to G13. A request r is randomly generated as follows: (1) The length of storage time interval $[a_r, t_r]$ uniformly ranges from 4 to 20 time points. (2) The arrival time interval between two consecutive requests follows a Poisson distribution with mean equal to 4. We assume that the arrival time of the first request a_1 is 1, and then obtain the arrival time points a_r of all requests. (3) The departure time t_r of a request cannot exceed 168 (the maximum planning horizon). (4) The total number of containers $n_{r,t}$ by time t is generated according to $\sum_{b\in B_r:a^b_r\le t}\sum_{t'\in[a^b_r,t]} n^{b,t'}_r$ where the size of set B_r is uniformly generated from 1 to 4. In order to simulate real cases, each element $b \in B_r$ represents a batch of containers arriving from the same origin, and containers from the same batch b are unloaded into the yard within a time interval $[a^b_r, t^b_r]$, where a^b_r and t^b_r are the time points when batch b starts and finishes unloading. We define $a_r = \min_{b\in B_r} a^b_r$ and $t_r = \max_{b\in B_r} t^b_r$. At each time point $t \in [a^b_r, t^b_r]$, there are $n^{b,t}_r$ arriving containers and $n^t_r = \sum_{b\in B_r} n^{b,t}_r$. The value of a^b_r, t^b_r and $n^{b,t}_r$ are generated as follows: (a) The length of interval $[a^b_r, t^b_r]$ is generated uniformly from 1 to the maximum feasible length ($t_r - a^b_r$) for batch b. (b) The arrival time points a^b_r between two consecutive batches follow Poisson distribution with a mean equal to 4 time units. (c) At each time point t with an arrival, the number of arriving containers $n^{b,t}_r$ follows an exponential distribution with a mean equal to 5 units. Before generating data, we define a metric, *load intensity (LI)*, to show the complexity of a instance. LI is defined as $LI = \frac{\sum_{r\in R}\sum_{i\in r} d_i}{L\times W\times H\times T}$ where d_i is the storage duration of container i of request r. For each instance group, we generate five instances with a different total number of containers and different load intensity.

As shown in Table 1, we have generated thirteen instance groups (G1~G13), classified into small, medium and large-scale groups based on the number of requests (N), the number of containers (Q) and load intensity (LI). The last Column "T" in Table 1 denotes the length of the planning horizon considered. A small-scale instance has less than 1800 containers from up to 25 requests. A medium-scale instance has around 1600–3100 containers from 30–45 requests, while a large-scale instance has around 2900–4300 containers from 50–65 requests.

[1] https://github.com/hitiantian/3dyapt.

Table 1
Test instance groups.

Scale	Group	N	Q (TEUs)	LI	T
Small	G1	5	100–400	1%–4%	168
	G2	10	400–800	4%–10%	168
	G3	15	700–1300	8%–15%	168
	G4	20	900–1500	9%–19%	168
	G5	25	1100–1800	11%–20%	168
Medium	G6	30	1600–2800	19%–33%	168
	G7	35	2000–2500	20%–35%	168
	G8	40	2200–2900	22%–36%	168
	G9	45	2700–3100	30%–34%	168
Large	G10	50	2900–3300	30%–40%	168
	G11	55	3400–3900	38%–43%	168
	G12	60	3400–3900	36%–46%	168
	G13	65	3600–4300	39%–49%	168

5.2. Lower bound

Since the problem objective is to find the minimal area of the rectangular cover s_C of all requests during the planning horizon T, a simple yet intuitive LB can be developed. The basic idea of the lower bound is to relax the non-decreasing constraints in the model, thus allowing containers in a request to be placed freely in the yard across time. With this relaxation, containers in different requests are to be placed closely adjacent to each other during each time unit, and the overall occupied storage space becomes extremely compact. Thus, this LB is calculated as the maximum sum of minimal storage space occupied by requests at each time among all time points during the planning horizon:

$$LB = \max_{0\le t\le|T|} \min_{s\in S_{r,t}} \sum_{r\in R_t} \Omega_{r,t},$$

where R_t is the set of requests that require storage space at time point t as defined in Section 2.2 and $\Omega_{r,t}$ is the area of storage space required by request r at time point t as defined in Section 3.1.

5.3. Comparison between SA-DSAP and SA-BL

Experiments are conducted on small, medium and large instances using both SA-DSAP and simulated annealing based bottom-left (SA-BL) (Fu et al., 2007). The values of initial yard lengths for the two algorithms are set based on different strategies. Yard length in the BL algorithm is set to a positive integer that is large enough to accommodate the containers required to be stored in the yard. It is also found that the value of yard length has little influence on the computational time required for BL algorithm. For the DSAP algorithm, the value of yard length is dynamically set and updated for each request following strategies introduced in Section 4.3. The computational time includes the time to generate the set $S_{r,t}$ ($r \in R, t \in T_r$).

Table 2 compares SA-DSAP and SA-BL on the small-scale instance groups. We can see that SA-DSAP significantly outperforms SA-BL. SA-DSAP obtains high-quality solutions whose values are only about 5% higher than the lower bounds. The averages of "Gap%" (=(z_2-LB)/LB) are 4.2%, 6.0%, 4.7%, 4.2% and 7.6% for G1 to G5 respectively. SA-DSAP also outperforms SA-BL by 43.2%, 37.3%, 33.1%, 35.4% and 41.3% for G1 to G5 respectively as shown by "Imp%" (=(z_1-z_2)/LB). Column "Time (s)" and "Iterations" in Table 2 give the total CPU runtime and the number of actual iterations executed in the simulated annealing procedure of both algorithms. It can be observed that SA-BL performs considerably faster than SA-DSAP in terms of CPU runtime. This is reasonable because SA-BL adopts a simple greedy strategy when allocating space to requests of containers while SA-DSAP employs a more time-consuming dynamic programming method. The planning time horizon is usually set to at least a week, so the computational time of half an hour is acceptable. From Column "Iterations", we can see that the number of iterations that SA has explored in both algorithms

Table 2
Comparison between SA-BL and SA-DSAP algorithms for small-scale instances.

Instances				SA-BL			SA-DSAP			Gap%	Imp%
Group	ID	LI	LB	z_1	Time (s)	Iterations	z_2	Time (s)	Iterations		
G1	5_1	1.2%	10	12	0.5	7200	10	26.3	7203	0.0%	20.0%
	5_2	2.1%	12	18	0.3	7935	12	0.1	7214	0.0%	50.0%
	5_3	3.5%	16	24	0.3	16845	18	197.8	15314	12.5%	37.5%
	5_4	3.3%	18	30	0.3	16875	18	0.4	15341	0.0%	66.7%
	5_5	2.8%	24	36	0.3	28114	26	209.2	23428	8.3%	41.7%
	Avg.									4.2%	43.2%
G2	10_1	4.1%	24	36	0.4	34628	26	252.7	31480	8.3%	41.7%
	10_2	6.3%	30	42	0.4	34962	30	12.0	31784	0.0%	40.0%
	10_3	6.2%	36	48	0.4	43762	39	536.7	39784	8.3%	25.0%
	10_4	8.6%	42	60	0.4	57941	44	668.1	48284	4.8%	38.1%
	10_5	9.8%	48	72	0.5	62902	52	600.4	57184	8.3%	41.7%
	Avg.									6.0%	37.3%
G3	15_1	7.6%	30	48	0.4	71729	30	332.7	65208	0.0%	60.0%
	15_2	12.8%	36	42	0.5	72265	36	34.1	65695	0.0%	16.7%
	15_3	11.1%	42	60	0.5	78010	45	578.5	74295	7.1%	35.7%
	15_4	13.4%	48	60	0.5	91515	54	583.5	83195	12.5%	12.5%
	15_5	14.2%	54	78	0.5	101305	56	971.4	92095	3.7%	40.7%
	Avg.									4.7%	33.1%
G4	20_1	18.1%	30	42	0.5	105043	30	176.5	93788	0.0%	40.0%
	20_2	16.0%	36	48	0.5	110982	39	948.9	102288	8.3%	25.0%
	20_3	9.0%	42	60	0.6	119094	44	1131.4	110888	4.8%	38.1%
	20_4	12.6%	48	66	0.6	132113	50	723.3	119776	4.2%	33.3%
	20_5	15.7%	54	78	0.6	154507	56	929.6	128745	3.7%	40.7%
	Avg.									4.2%	35.4%
G5	25_1	11.5%	30	48	0.6	153266	33	930.8	136845	10.0%	50.0%
	25_2	11.6%	36	54	0.6	157801	39	795.4	145439	8.3%	41.7%
	25_3	17.7%	42	54	0.6	156217	42	2.4	145453	0.0%	28.6%
	25_4	18.2%	48	72	0.6	169738	52	1111.5	153888	8.3%	41.7%
	25_5	20.5%	54	84	0.8	163341	60	1330.2	162804	11.1%	44.4%
	Avg.									7.6%	41.3%

Table 3
Comparison between SA-BL and SA-DSAP algorithms for medium-scale instances.

Instances				SA-BL			SA-DSAP			Gap%	Imp%
Group	ID	LI	LB	z_1	Time (s)	Iterations	z_2	Time (s)	Iterations		
G6	30_1	19.8%	36	48	0.7	183742	39	1558.0	171241	8.3%	25.0%
	30_2	19.2%	42	60	0.8	195738	48	1286.1	179741	14.3%	28.6%
	30_3	19.7%	48	66	0.8	192414	50	908.9	188641	4.2%	33.3%
	30_4	21.8%	54	66	0.8	240514	56	1450.8	197143	3.7%	18.5%
	30_5	33.2%	60	90	1.0	223006	65	1800.2	205915	8.3%	41.7%
	Avg.									7.8%	29.4%
G7	35_1	20.6%	40	60	0.8	245580	42	1701.3	214335	5.0%	45.0%
	35_2	21.7%	48	66	0.9	254640	52	1800.0	222807	8.3%	29.2%
	35_3	24.9%	54	72	1.0	265987	60	998.6	231763	11.1%	22.2%
	35_4	27.4%	60	84	1.1	275033	65	1800.1	240638	8.3%	31.7%
	35_5	33.1%	66	90	1.1	284946	70	1718.7	249730	6.1%	30.3%
	Avg.									7.8%	31.7%
G8	40_1	21.8%	36	54	1.0	296998	39	1215.1	258262	8.3%	41.7%
	40_2	30.9%	42	72	1.0	304466	44	1800.2	265334	4.8%	66.7%
	40_3	24.2%	48	72	0.9	314062	52	1680.1	273934	8.3%	41.7%
	40_4	35.2%	54	78	1.2	324753	65	1749.1	282847	20.4%	24.1%
	40_5	24.6%	60	84	1.0	334121	65	1469.7	291846	8.3%	31.7%
	Avg.									10.0%	41.1%
G9	45_1	30.0%	42	54	1.2	344358	48	1661.7	300546	14.3%	14.3%
	45_2	31.3%	48	72	1.2	353040	52	1800.1	308906	8.3%	41.7%
	45_3	33.8%	54	84	1.4	364649	60	1792.3	317730	11.1%	44.4%
	45_4	31.6%	60	90	1.3	373350	66	1543.4	326660	10.0%	40.0%
	45_5	29.9%	66	84	1.2	383047	72	1316.7	335707	9.1%	18.2%
	Avg.									10.6%	31.7%

Table 4
Comparison between SA-BL and SA-DSAP algorithms for large-scale instances.

Instances				SA-BL			SA-DSAP			Gap%	Imp%
Group	ID	LI	LB	z_1	Time (s)	Iterations	z_2	Time (s)	Iterations		
G10	50_1	35.7%	48	66	1.4	393857	60	1800.0	342488	25.0%	12.5%
	50_2	29.9%	54	78	1.3	403212	56	1779.9	351389	3.7%	40.7%
	50_3	34.9%	60	90	1.4	413494	70	1540.7	360662	16.7%	33.3%
	50_4	38.3%	66	90	1.6	422555	72	1800.1	368029	9.1%	27.3%
	50_5	39.1%	72	96†	1.5	432009	75	1507.0	377348	4.2%	29.2%
	Avg.									11.7%	28.6%
G11	55_1	38.1%	48	72	1.6	439437	60	1800.1	383528	25.0%	25.0%
	55_2	42.5%	54	84	1.6	447513	65	1800.2	391569	20.4%	35.2%
	55_3	38.0%	60	84	1.5	459767	66	1800.1	400610	10.0%	30.0%
	55_4	40.4%	66	96†	1.7	468511	75	1553.9	409921	13.6%	31.8%
	55_5	42.5%	72	90	1.6	479327	78	1641.1	420088	8.3%	16.7%
	Avg.									15.5%	27.7%
G12	60_1	36.2%	48	66	1.5	492881	54	1383.2	428597	12.5%	25.0%
	60_2	42.7%	54	90	1.8	502272	66	1789.7	437717	22.2%	44.4%
	60_3	39.1%	60	90	1.6	511482	66	1800.2	446130	10.0%	40.0%
	60_4	40.5%	66	78	1.8	520919	70	1800.2	453700	6.1%	12.1%
	60_5	45.2%	72	108†	1.8	530219	78	1800.0	463132	8.3%	41.7%
	Avg.									11.8%	32.6%
G13	65_1	39.3%	55	72	1.7	539319	66	1800.2	470702	20.0%	10.9%
	65_2	49.0%	60	84	1.8	548775	70	1677.0	480172	16.7%	23.3%
	65_3	45.8%	66	102†	1.9	558017	72	1800.1	486218	9.1%	45.5%
	65_4	38.8%	72	102†	1.9	567417	84	1800.1	496458	16.7%	25.0%
	65_5	41.5%	78	126†	2.0	576687	84	1800.0	505415	7.7%	53.8%
	Avg.									14.0%	31.7%

Note: Objective value marked by † exceeds maximum yard area Ω_{max} (=90) and is infeasible.

are comparable. Besides, it can seen that some solutions by SA-DSAP equal to the LBs in Table 2. This implies that there is greater chance for instances with low load intensity LI to reach optimal solutions. The reasons are from two aspects. First, the lower bound is calculated based on the relaxation of the non-decreasing constraints, so a solution simply adopts the most compact way for placement of containers from different requests independently at each time point. Second, for small-scale instances in Table 2, when the "Load Intensity" is low, the yard space becomes sufficiently large and hence the requests can be easily placed in the most compact way when the storage space is allocated. As a result, for the small-scale instances with low "Load Intensity", objective values of the feasible solutions are more likely to be equal to the lower bounds.

Table 3 illustrates the comparison between SA-DSAP and SA-BL on the medium-scale instance groups (G6~G9). The average values of "Imp%" of the four groups are 29.4%, 31.7%, 41.1%, and 31.7% respectively. The average of "Gap%" is around 10% for medium-scale instances, which indicates that the results of SA-DSAP are close to the estimated lower bounds.

Table 4 shows comparison between SA-DSAP and SA-BL on the large-scale groups (G10~G13). It can be observed that all the twenty solution values by SA-DSAP (Column 2) are within the preset maximum yard area Ω_{max} ($W \times L = 90$) while six of twenty solutions by SA-BL

Table 5
Comparison between solutions by SA-DSAP and CPLEX.

Group	ID	LB	CPLEX (**M2**)		SA-DSAP		Diff%
			z_1	Time (s)	z_2	Time (s)	
G1	5_1	10	**10**	0.76	**10**	0.25	0.00%
	5_2	12	**12**	0.43	**12**	0.23	0.00%
	5_3	16	**18**	1.50	**18**	0.93	0.00%
	5_4	18	**18**	1.03	**18**	0.91	0.00%
	5_5	24	**26**	12.92	**26**	0.24	0.00%
G2	10_1	24	**26**	85.08	**26**	3.13	0.00%
	10_2	30	**30**	11.89	**30**	4.20	0.00%
	10_3	36	**39**	61.98	**39**	0.08	0.00%
	10_4	42	**44**	438.47	**44**	1.27	0.00%
	10_5	48	**52**	763.48	**52**	8.52	0.00%
G3	15_1	30	**30**	38.08	**30**	0.71	0.00%
	15_2	36	**36**	44.23	**36**	13.65	0.00%
	15_3	42	**45**	149.78	**45**	2.44	0.00%
	15_4	48	**52**	719.91	**52**	93.68	0.00%
	15_5	54	**56**	599.09	**56**	32.39	0.00%
G4	20_1	30	**30**	62.16	**30**	54.14	0.00%
	20_2	36	**39**	594.15	**39**	20.58	0.00%
	20_3	42	**44**	1977.01	**44**	378.91	0.00%
	20_4	48	**50**	34691.52	**50**	76.22	0.00%
	20_5	54	**55**	2370.67	56	180.07	1.82%
G5	25_1	30	**33**	628.40	**33**	0.14	0.00%
	25_2	36	**39**	371.49	**39**	79.11	0.00%
	25_3	42	**42**	104.77	**42**	11.24	0.00%
	25_4	48	**52**	14631.70	**52**	34.26	0.00%
	25_5	54	60	86400.00	60	168.23	0.00%

Table 6
Results on "container number per request" factor.

ID	per	z	Delta	ID	Per	z	Delta	ID	Per	z	Delta
10_4	0%	44	0	30_2	0%	48	0	50_2	0%	56	0
10_4_5	5%	51	7	30_2_5	5%	55	7	50_2_5	5%	70	14
10_4_10	10%	51	7	30_2_10	10%	55	7	50_2_10	10%	70	14
10_4_15	15%	54	10	30_2_15	15%	60	12	50_2_15	15%	72	16
10_4_20	20%	55	11	30_2_20	20%	60	12	50_2_20	20%	75	19
10_4_25	25%	60	16	30_2_25	25%	60	12	50_2_25	25%	78	22
10_4_30	30%	60	16	30_2_30	30%	65	17	50_2_30	30%	84	28
10_4_35	35%	63	19	30_2_35	35%	70	22	50_2_35	35%	84	28
10_4_40	40%	63	19	30_2_40	40%	70	22	50_2_40	40%	90	34
10_4_45	45%	66	22	30_2_45	45%	70	22	50_2_45	45%	90	34
10_4_50	50%	66	22	30_2_50	50%	70	22	50_2_50	50%	90	34

exceed this limit of 90. This suggests that SA-DSAP can cope with the situation well when high usage of yard space is expected. From Column "Gap%", it can be found that SA-DSAP obtains high-quality solutions whose values are only about 15% higher than the lower bounds. The averages of "Gap%" are 11.7%, 15.5%, 11.8%, and 14.0% for G10 to G13 respectively. Column "Imp%" shows that SA-DSAP outperforms SA-BL by 28.6%, 27.7%, 32.6% and 31.7% respectively. In general, SA-DSAP is capable of finding high-quality storage solutions when facing a large number of requests involving a large number of containers during the planning horizon.

5.4. Comparison with optimal solutions on small-scale instances

In this subsection, we conduct additional computational experiments to verify the effectiveness of our approach. SA-DSAP is benchmarked on small-scale instances (G1∼G5) solvable to optimality based on a reduced ILP model (**M2**). The reduced ILP model (**M2**) is developed based on the original ILP model discussed in Section 2.2 and solutions obtained in Section 5.3; we incorporate two additional constraints:

$$\pi_p = 0, \text{ if } x_p y_p < \text{LB or } x_p y_p > \text{UB}, \forall p \in P, \tag{15}$$

$$\lambda_{r,t,s} = 0, \text{ if } x_s^+ y_s^+ > \text{UB}, \forall s \in S_{r,t}, \forall r \in R, \forall t \in T_r, \tag{16}$$

where LB is the estimated lower bound (Section 5.2) and UB is the feasible solution, also an upper bound, obtained by SA-DSAP. As we find out, the solution space in M2 has been greatly reduced after introduction of lower and upper bounds, thus making the model solvable by CPLEX, though it may take more than 24 h to get the optimal solution for a single instance.

In Table 5, Column "CPLEX (**M2**)" shows the optimal objective values solved by CPLEX. Column "SA-DSAP" shows the objective values by SA-DSAP within 30 min. Column "Diff%" ($=(z_2 - z_1)/z_1$) shows the gap between the solutions by SA-DSAP to the solutions by the general optimization solver CPLEX.

From Table 5, we observe that SA-DSAP finds the optimal solution in 23 out of 25 instances. For instance 20_5, the result of SA-DSAP is only slightly worse than the solved optimum by 1 unit (1.82%) in yard length, as shown in Column "Diff%". While the computing times for the already reduced ILP model (**M2**) are very long, requiring up to a few days using CPLEX, while SA-DSAP takes less than a few minutes. For instance 25_5 where CPLEX does not reach optimal, the objective value of SA-DSAP is equal to that of the feasible solution obtained by **M2** with much shorter computational time. The overall results of the experiments are encouraging, since high-quality solutions can be generated within a short period of time by SA-DSAP.

5.5. Sensitivity analysis

In this subsection, we conduct a set of sensitivity analysis. We first test the effect of container quantity per request on storage solutions.

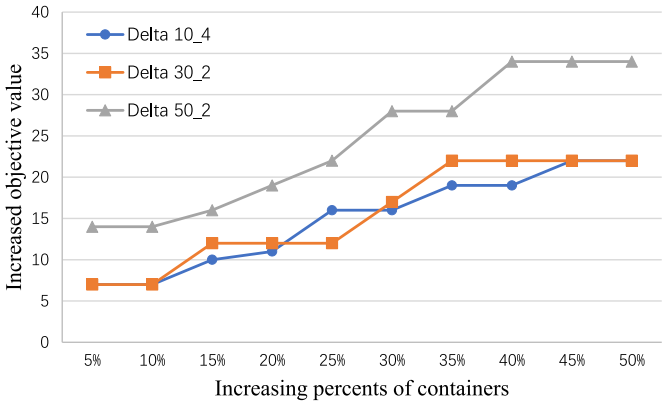


Fig. 6. Illustration of results when container number increases.

Table 7
Results on "yard length" factor.

ID	L	z	ID	L	z	ID	L	z
	20	44		20	48		20	56
	19	44		19	48		19	56
	18	44		18	48		18	56
	17	44		17	48		17	56
	16	44		16	48		16	60
10_4	15	44	30_2	15	48	50_2	15	60
	14	44		14	48		14	60
	13	44		13	48		13	60
	12	44		12	48		12	60
	11	44		11	48		11	60
	10	45		10	48		10	60

We select three instances from the small, medium and large-scale data, 10_4, 30_2, 50_2. For each instance, we increase the number of containers by 5% to 50% and keep the storage time of the containers unchanged, resulting in 10 new instances as a group. We then apply the SA-DSAP algorithm on these three instance groups. The results are listed in Table 6 as follows.

In Table 6, Column "ID", Column "per" and Column "z" represents the case name, increase in the number of containers and the objective function value, respectively. Column "Delta" represents the increase of objective value as compared with the original objective value. From Table 2, we can see that in general the more containers a request needs to process, the more space required (see Fig. 6), but for some instances such as 10_4_45 to 10_4_50, the overall space required does not change. The reason for the latter case, as we find out, is that there is much unused space in the original solution due to an uneven distribution of containers in the requests, and thus with the increase in the number of containers per request, the unused space is occupied first and the objective value does not change.

We then test the effect of yard length on the solutions. The current yard has a length of 20 units, and we reduce this yard length and keep other parameters unchanged. We select three instances, small, medium and large-scale from data, i.e., 10_4, 30_2, 50_2, and decrease the yard length by one unit each time from 20 to 10, resulting in a group of 10 new instances for each selected instance. We apply SA-DSAP on these new instance groups and present the results in Table 7.

In Table 7, Column "ID", Column "L" and Column "z" represents the case name, the yard length and the objective function value, respectively. From Table 7, we observe that for small and medium-scale instances (10_4 and 30_2), the solution remains almost the same as the yard length decreases. For large-scale instances (50_2), the solution starts to change when the yard length decreases by a significant value (in this case decreased by 4 units to 16), as shown in Fig. 7. The reason for this is that there is a large quantity of containers and there is a significant decrease in the yard length, the flexibility in placing

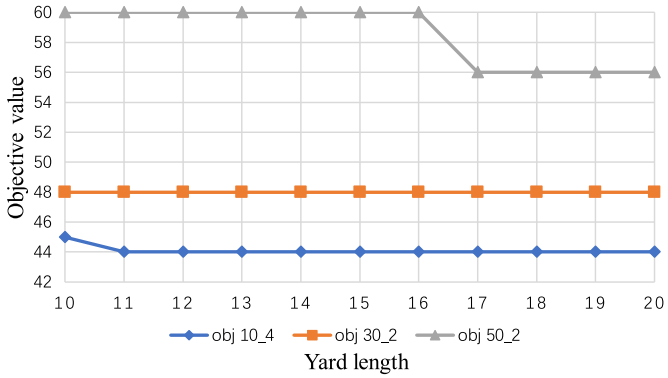


Fig. 7. Illustration of results when yard length increases.

Table 8
Results on "storage space shape" factor.

ID	d	z	d	z	d	z	d	z
10_4	0	60	4	45	8	44	12	44
	1	48	5	44	9	44	13	44
	2	48	6	44	10	44	14	44
	3	45	7	44	11	44	15	44

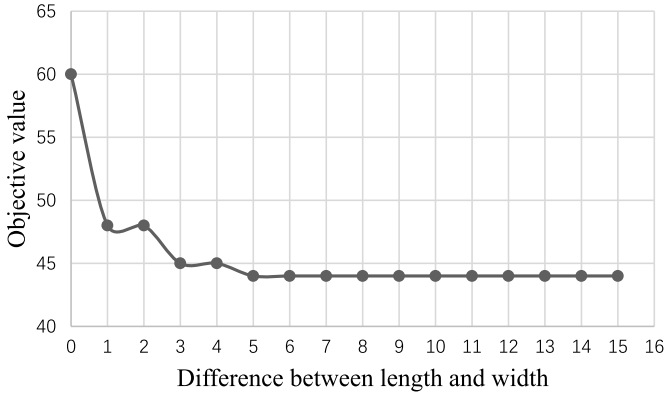


Fig. 8. Illustration of the effect of d-value.

containers in all the requests in the available yard space also decreases significantly. Therefore, the objective values obtained by the proposed SA-DSAP algorithm become larger.

Finally, we test the effect of the difference d in length and width of the storage space on the solutions. We take a small-scale instance 10_4 and change the value of d from 0 to 15. SA-DSAP is then applied to this group of 16 instances. The results are presented in Table 8.

In Table 8, Column "ID", Column "d" and Column "z" represents the case name, the value of the difference d and the objective function value, respectively. As we observe from the table, for the square storage space ($d = 0$), our results are consistent with Fu et al. (2007). As the d-value increases, the objective value decreases initially and then remains unchanged. This implies that SA-DSAP algorithms are good at finding efficient storage solutions when there is more flexibility (larger d-values) in the shape of storage space. However, when the d-value exceeds a certain limit, the advantage of this flexibility appears less significant (Fig. 8).

5.6. Improvement on space utilization by SA-DSAP

To compare the unused space in solutions of SA-DSAP and SA-BL, we define $ratio_{unused}$ to indicate the proportion of unused sparse storage

Table 9
Results of unused space ratio on small, medium and large-scale data.

ID	T_s	SA-BL			SA-DSAP			Imp (%)
		z	$area_{total}$	$ratio_{unused}$ (%)	z	$area_{total}$	$ratio_{unused}$(%)	
15_1	67	48	870	73%	30	801	60%	13%
15_2	95	42	1456	64%	36	1400	59%	4%
15_3	98	60	1573	73%	45	1478	66%	7%
15_4	57	60	1527	55%	54	1449	53%	2%
15_5	69	78	1702	68%	56	1598	59%	10%
35_1	150	60	2672	70%	42	2506	60%	10%
35_2	156	66	2742	73%	52	2559	68%	5%
35_3	117	72	2878	66%	60	2734	61%	5%
35_4	133	84	3574	68%	65	3295	62%	6%
35_5	149	90	3823	71%	70	3654	65%	7%
55_1	165	72	4569	62%	60	4418	55%	6%
55_2	162	84	5195	62%	65	5057	52%	10%
55_3	157	84	4004	70%	66	3835	63%	7%
55_4	162	96	4794	69%	70	4621	59%	10%
55_5	165	90	4885	67%	78	4507	65%	2%

Table 10
Comparison results on initial yard status.

ID	z^E	z^{NE}	Delta	ID	z^E	z^{NE}	Delta	ID	z^E	z^{NE}	Delta
20_1	30	30	0	40_1	39	40	1	60_1	54	54	0
20_2	39	39	0	40_2	44	45	1	60_2	66	66	0
20_3	44	46	2	40_3	52	52	0	60_3	66	66	0
20_4	50	54	4	40_4	65	65	0	60_4	70	70	0
20_5	56	60	4	40_5	65	65	0	60_5	78	80	2

Table 11
Results of threshold LI values based on instance group G13 (size = 65)

65_1		65_2		65_3		65_4		65_5	
LI	z	LI	z	LI	z	LI	z	LI	z
50.28%	84	55.84%	88	55.34%	90	46.66%	88	49.73%	90
52.12%	88	56.33%	90	57.23%	92	54.47%	100	57.99%	100
55.07%	92	56.82%	87	58.83%	90	62.28%	112	66.26%	110
57.78%	96	57.31%	96	60.78%	102	70.10%	128	74.52%	115
58.42%	96	57.80%	96	64.12%	102	77.91%	144	82.78%	132

spaces in a solution **s**, as follows:

$$ratio_{unused} = 1 - \frac{area_{\mathbf{s}}}{z_{\mathbf{s}} \times |T_{\mathbf{s}}|},$$

where $area_{\mathbf{s}}$ is the total storage space used in solution **s**. $z_{\mathbf{s}}$ is the objective value of solution **s**, i.e., the area of the rectangular cover in solution **s**. $|T_{\mathbf{s}}|$ is the length of time for which containers are stored in the yard during the planning horizon. Computational experiments are conducted for both SA-DSAP and SA-BL on instance groups with size 15, 35 and 55 (small, medium and large-scale instances). The results are reported in the following Table 9.

As shown by the last column in Table 9, the unused space ratio $ratio_{unused}$ has been improved significantly by SA-DSAP compared with SA-BL. For small-scale instances, the improvement of $ratio_{unused}$ is between 2% and 13%. For medium-scale instances, the improvement of $ratio_{unused}$ is between 5% and 10%. For large-scale instances, the improvement of $ratio_{unused}$ is between 2% and 10%. We are then able to conclude that SA-DSAP can generate more efficient yard allocation solutions with less unused space compared with SA-BL.

5.7. Non-empty yard

To test the effectiveness of the SA-DSAP algorithm in solving these cases with initially non-empty yards, we select three instance groups G4, G8, G12 (size=20, 40, 60) from small-scale to medium-scale and large-scale instances. For each instance, we randomly generate a non-empty initial yard status, that is, some requests are in the middle of being processed in the yard at the beginning of the planning time horizon, and therefore some portion of yard space has already been occupied. All these instances are solved by SA-DSAP. Results are shown in Table 10.

In Table 10, z^E and z^{NE} represent the objective function value obtained from the initially empty instances and the initially non-empty instances. Delta is defined as $z^{NE} - z^E$. The results show that even if the yard is initially non-empty, our proposed algorithm can still obtain very competitive results in terms of yard usage that do not differ very much from the initial empty instances.

5.8. Effects on load intensity

In this subsection, we conduct experiments to analyze how the SA-DSAP algorithm is affected by the load intensity (LI) of an instance. In order to find the threshold LI value that SA-DSAP cannot find a feasible solution, we gradually increase LI for a few instances, i.e., 61_1 to 65_5 from the largest data set G13 in Table 11 and generate five new data sets. We then apply the SA-DSAP algorithm to all instances.

In Table 11, Column "LI" and Column "z" represent the "Load Intensity" and the objective values. It can be seen that for the 65_1 instance, when LI increases to 55.07%, the SA-DSAP algorithm cannot obtain a feasible solution. Similarly, threshold LI values for instances 65_2, 65_3, 65_4 and 65_5 are found to be 57.31%, 57.23%, 54.47%, and 57.99% respectively. Therefore, it is observed from the experiments that there is a great chance of the SA-DSAP algorithm being unable to find a feasible solution for an instance in the largest data set G13 when the load intensity is above 50%.

In order to further ascertain how the LI affects the performance of SA-DSAP as well as the feasibility of the instances, we generate three more groups of large instances, G14-G16. As shown in Table 12, each

Table 12
Results of SA-DSAP on large instances with high load intensities.

Group	ID	LI	LB	Gap_{max}	z	Time (s)
G14	75_1	50.80%	81	10%	88	1733.26
	75_2	51.09%	83	8%	90	396.19
	75_3	52.90%	85	6%	96[a]	1517.79
	75_4	54.38%	87	3%	100[a]	67.388
	75_5	52.30%	88	2%	108[a]	251.123
G15	80_1	52.99%	83	8%	102[a]	774.755
	80_2	55.38%	85	6%	96[a]	960.078
	80_3	55.23%	87	3%	100[a]	135.526
	80_4	52.86%	88	2%	102[a]	358.257
	80_5	55.51%	89	1%	100[a]	22.892
G16	85_1	53.41%	84	7%	95[a]	18.11
	85_2	55.84%	82	9%	102[a]	1.306
	85_3	53.29%	85	6%	108[a]	24.76
	85_4	56.75%	89	1%	114[a]	29.551
	85_5	57.74%	90	0%	108[a]	805.004

[a]Note: Objective values exceeds maximum yard area Ω_{max} (=90) and is infeasible.

group consists of five instances with load intensities (Column "LI") greater than 50%. It can be observed that the lower bounds (Column "LB") obtained by relaxing the non-decreasing constraints in the model are quite close to the maximum yard area (Ω_{max}) of 90. The gaps between the lower bounds and the maximum yard area Ω_{max} (Column "Gap_{max}"$=\frac{\Omega_{max}-LB}{LB} \times 100\%$) are all within 10%, which indicates that load intensity above 50% is very high and is likely to result in no feasible solution being found. Computational results show that the SA-DSAP is only able to find feasible solutions ($z \leq \Omega_{max} = 90$) for instance 75_1 and 75_2 in instance group G14. No feasible solution is obtained for the remaining 13 large instances with high load intensities.

6. Conclusion

In this paper, we study the 3D yard allocation problem with time dimension (3DYAPT) at the operational level. Containers in each request are stacked in the yard following the consignment strategy. We aim to minimize the area of a rectangular yard space ever occupied by all requests within the planning horizon while satisfying non-decreasing constraints and the non-overlapping requirements. An integer linear programming model is formulated for the problem, but it cannot be solved directly especially for medium-scale and large-scale instances due to NP-hardness of the 3DYAPT.

We propose a SA-DSAP algorithm that dynamically adjusts the shape of the stack of containers for each request at each time point. To balance the search time of the DSAP algorithm and the solution quality, a customized initial yard length strategy is proposed for each request based on both the information of the request in process and the instant layout information of the yard. Our work has its merit in practice because computational experiments show that the SA-DSAP algorithm outperforms the algorithm proposed by Fu et al. (2007) by 27.7% to 32.6% for large-scale groups based on average values, which means marked economic implications for ports. Besides, the SA-DSAP obtains optimal solutions within a short time compared with the general optimization solver CPLEX. Therefore, the proposed SA-DSAP algorithm is effective for solving the 3DYAPT.

As for future study, there are several directions in which our work can be further improved. First, it is possible to extend the 3DYAPT by modifying constraints or objectives to address new practical aspects or requirements. For example, the rectangular storage shape assumption could be relaxed. Storage spaces with irregular boundaries, such as the flexible bay boundary shared by two neighboring segments (Zhou et al., 2020), can be studied. Constraints that limit container tier difference can be considered. It is possible to incorporate this requirement by adding a post-processing procedure to the SA-DSAP algorithm that adjusts the height of container tiers based on a given strategy. Other requirements such that containers should be piled along the yard crane moving direction or in the length dimension, coupled with "container reshuffling", during the whole planning horizon can also be studied. The problem itself can also be extended to integrate yard space allocation with crane scheduling, thus taking reduction of energy consumption of crane operations (Iris and Lam, 2019) as one of the dual objectives into consideration. Second, there is currently still room for improvement in the solution approach. The proposed SA-DSAP algorithm is a heuristic method, in which requests are evaluated one by one according a pre-assigned priority sequence. Extensions can be made to consider allocation of two or more requests together in the dynamic programming in order to make the algorithm more efficient. For some simple or restricted cases of the 3DYAPT, it is also possible to develop approximation algorithms with certain performance guarantee.

CRediT authorship contribution statement

Tiantian Wang: Conceptualization, Investigation, Algorithm design and implementation, Writing – original draft. **Hong Ma:** Investigation, Methodology, Writing – review & editing, Supervision. **Zhou Xu:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Jun Xia:** Methodology, Writing – review & editing.

Acknowledgment

The authors would like to express their thanks to all four anonymous referees whose comments greatly helped improve the presentation of this work. This research was supported in part by the National Natural Science Foundation of China [Grant no. 71821002; 71201141; 71831008].

References

Bazzazi, M., Safaei, N., Javadian, N., 2009. A genetic algorithm to solve the storage space allocation problem in a container terminal. Comput. Ind. Eng. 56 (1), 44–52.
Buhrkal, K., Zuglian, S., Ropke, S., Larsen, J., Lusby, R., 2011. Models for the discrete berth allocation problem: A computational comparison. Transp. Res. E 47 (4), 461–473.
Chen, P., Fu, Z., Lim, A., 2002. The yard allocation problem. In: AAAI/IAAI. pp. 3–8.
Fu, Z., Li, Y., Lim, A., Rodrigues, B., 2007. Port space allocation with a time dimension. J. Oper. Res. Soc. 58 (6), 797–807.
Gharehgozli, A., Zaerpour, N., 2018. Stacking outbound barge containers in an automated deep-sea terminal. European J. Oper. Res. 267 (3), 977–995.
Iris, Ç., Christensen, J., Pacino, D., Ropke, S., 2018. Flexible ship loading problem with transfer vehicle assignment and scheduling. Transp. Res. B 111, 113–134.
Iris, Ç., Lam, J.S.L., 2019. A review of energy efficiency in ports: Operational strategies, technologies and energy management systems. Renew. Sustain. Energy Rev. 112, 170–182.
Iris, Ç., Pacino, D., Ropke, S., Larsen, A., 2015. Integrated berth allocation and quay crane assignment problem: Set partitioning models and computational results. Transp. Res. E 81, 75–97.

Jiang, X.J., Jin, J.G., 2017. A branch-and-price method for integrated yard crane deployment and container allocation in transshipment yards. Transp. Res. B 98, 62–75.
Jin, J.G., Lee, D.-H., Cao, J.X., 2016. Storage yard management in maritime container terminals. Transp. Sci. 50 (4), 1300–1313.
Kang, J., Ryu, K.R., Kim, K.H., 2006. Deriving stacking strategies for export containers with uncertain weight information. J. Intell. Manuf. 17 (4), 399–410.
Kim, K.H., Park, Y.M., Ryu, K.-R., 2000. Deriving decision rules to locate export containers in container yards. European J. Oper. Res. 124 (1), 89–101.
Lee, D.-H., Jin, J.G., 2013. Feeder vessel management at container transshipment terminals. Transp. Res. E 49 (1), 201–216.
Lee, D.-H., Jin, J.G., Chen, J.H., 2011. Integrated bay allocation and yard crane scheduling problem for transshipment containers. Transp. Res. Rec. 2222 (1), 63–71.
Lee, D.-H., Jin, J.G., Chen, J.H., 2012. Terminal and yard allocation problem for a container transshipment hub with multiple terminals. Transp. Res. E 48 (2), 516–528.
Lim, A., Xu, Z., 2006. A critical-shaking neighborhood search for the yard allocation problem. European J. Oper. Res. 174 (2), 1247–1259.
Liu, C., 2020. Iterative heuristic for simultaneous allocations of berths, quay cranes, and yards under practical situations. Transp. Res. E 133, 101814.
Martello, S., Pisinger, D., Vigo, D., 2000. The three-dimensional bin packing problem. Oper. Res. 48 (2), 256–267.
Moccia, L., Cordeau, J.-F., Monaco, M.F., Sammarra, M., 2009. A column generation heuristic for a dynamic generalized assignment problem. Comput. Oper. Res. 36 (9), 2670–2681.
Silva, E.F., Wauters, T., et al., 2019. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. Comput. Oper. Res. 109, 12–27.
Tan, C., He, J., Wang, Y., 2017. Storage yard management based on flexible yard template in container terminal. Adv. Eng. Inf. 34, 101–113.
Tao, Y., Lee, C.-Y., 2015. Joint planning of berth and yard allocation in transshipment terminals using multi-cluster stacking strategy. Transp. Res. E 83, 34–50.
Ting, C.-J., Wu, K.-C., 2017. Optimizing container relocation operations at container yards with beam search. Transp. Res. E 103, 17–31.
Wei, L., Oon, W.-C., Zhu, W., Lim, A., 2011. A skyline heuristic for the 2D rectangular packing and strip packing problems. European J. Oper. Res. 215 (2), 337–346.
Wu, Y., Li, W., Goh, M., de Souza, R., 2010. Three-dimensional bin packing problem with variable bin height. European J. Oper. Res. 202 (2), 347–355.
Zhang, C., Liu, J., Wan, Y.-w., Murty, K.G., Linn, R.J., 2003. Storage space allocation in container terminals. Transp. Res. B 37 (10), 883–903.
Zhen, L., Chew, E.P., Lee, L.H., 2011. An integrated model for berth template and yard template planning in transshipment hubs. Transp. Sci. 45 (4), 483–504.
Zhen, L., Xu, Z., Wang, K., Ding, Y., 2016. Multi-period yard template planning in container terminals. Transp. Res. B 93, 700–719.
Zhou, C., Wang, W., Li, H., 2020. Container reshuffling considered space allocation problem in container terminals. Transp. Res. E 136, 101869.