European Journal of Operational Research 327 (2025) 84–94

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Iterated greedy for the yard crane scheduling problem with input/output assignment

Hongtao Wang [a,*], Rubén Ruiz [a,b], Fulgencia Villa [a], Eva Vallada [a,1]

[a] Grupo de Sistemas de Optimización Aplicada, Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain
[b] Amazon.com, Inc. P.O. Box 81226, Seattle, WA, United States

ARTICLE INFO

Keywords:
Iterated Greedy
Yard crane scheduling problem
Heuristics
Input/output assignment
Terminal planning

ABSTRACT

The yard crane scheduling problem (YCSP) consists of optimizing container loading for storage and retrieval requests from yard cranes at port terminals. This paper studies a realistic generalization of the YCSP that incorporates the assignments of input/output (I/O) points during the optimization stage. I/O points serve as buffers between the different transportation modes in the port terminal. These are limited in number and unproductive idle times might result if a container schedule exhausts I/O point availability. The resulting problem entails not only scheduling container storage and retrieval requests, but also the assignment of the I/O points. We introduce a series of simple, yet powerful, Iterated Greedy (IG) methods. These include variations of the destruction and reconstruction operators, coordination with novel local search procedures and problem-specific knowledge speed-ups. The proposed IG methods are carefully calibrated and evaluated using comprehensive computational experiments. The results indicate that small changes in the features of the algorithm have a profound impact on performance. Comparisons against the state-of-the-art approaches for this particular problem result in a strong, and statistically significant performance advantage for the proposed IG procedures.

1. Introduction

Containerized transport is a major factor in modern trade. The world's container port traffic reached 852.3 million Twenty-feet Equivalent Units (TEUs) in 2022 after increasing 18-fold from 1980, making it the fastest-growing segment of maritime transport (UNCTAD, 2023). This rapid growth has put enormous pressure on port terminals and this is particularly significant in the case of yard cranes (YCs). There is a challenge in the effective deployment of YCs in yard terminals as they are usually bulky, slow and need to move frequently between different locations (Li et al., 2009; Ng & Mak, 2005a). In order to keep pace with increased demand, it is critical to optimize the loading process of containers in terminals. Yard crane scheduling problems (YCSPs) are formulated and studied precisely with this aim in mind.

Scheduling problems concerning the YC deal with how to arrange the delivery of containers loaded by the YC in terminals. Fig. 1 displays a block of a classic European terminal layout with 6 Rows, 9 Bays, 3 Tiers, Quay Cranes (QCs), YCs, Automated Guided Vehicle (AGVs), Straddle Carriers (SCs) and 4 Input/Output (I/O) points at each side. Using the vehicles (AGVs and trucks), containers are transported to destinations for different container types. Storage/inbound containers are delivered from the outside to the landside or seaside and are stored in yard blocks. Retrieval/outbound containers are already stacked in yard blocks and are sent to the landside or seaside. All storage and retrieval containers must be loaded by the YCs between the yard block and the I/O points at the corresponding sides based on their destinations. The I/O points decouple the transfer between the YCs and the vehicles and potentially result in gains of high efficiency. However, sequencing of the operations of the YCs must observe the times at which containers are expected to be picked up at the I/O points. Otherwise, productivity gains might be nullified by the waiting times of the containers at the I/O points or even by I/O point congestion.

From the initial work of Ng and Mak (2005a, 2005b), the YCSP has received increased attention. Ng and Mak (2005a) formulated the YCSP as a machine scheduling problem to minimize the total waiting time and also the total time cost in Ng and Mak (2005b) where the time needed to finish a job is always fixed and known in advance. In the case of a single YC (machine), the whole schedule can be seen as the traveling route of the crane. Therefore, YCSPs have often been modeled as routing problems (Gharehgozli et al., 2014; Vallada et al., 2023; Vis & Roodbergen, 2009). In these studies, problems with

* Corresponding author.
E-mail addresses: hwang8@doctor.upv.es (H. Wang), rruiz@eio.upv.es (R. Ruiz), mfuvilju@eio.upv.es (F. Villa), evallada@eio.upv.es (E. Vallada).
[1] Senior Research Fellow at ValgrAI.

https://doi.org/10.1016/j.ejor.2025.04.052
Received 9 November 2024; Accepted 29 April 2025
Available online 23 May 2025

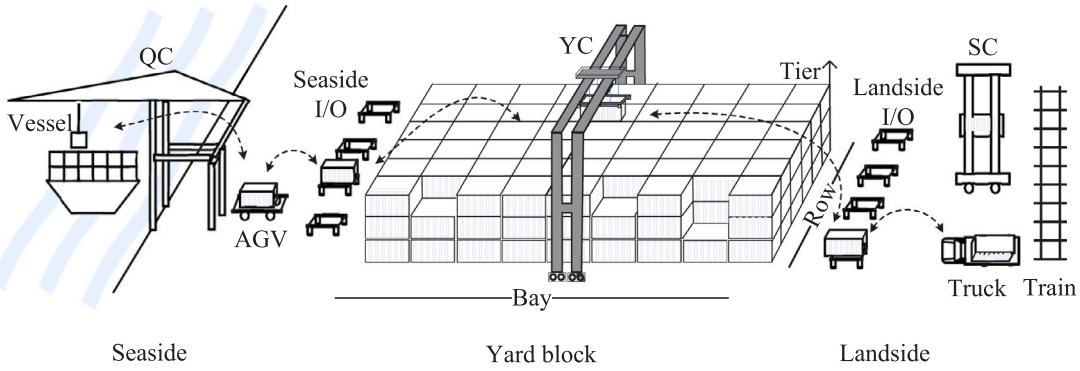

Fig. 1. Container transfer process involving different equipment like a quay crane (QC), Automated Guided Vehicles (AGVs), a yard crane (YC) and straddle carrier (SC) in a typical container terminal with a European layout.

the loading process are often addressed, where two important ones distinguish the YCSP from previous formulations. Vis and Roodbergen (2009) considered retrieval and storage requests along with the origins and destinations on different sides (sea and land). Gharehgozli et al. (2014) introduced the I/O points for the first time to receive and send containers between the yard block and different sides. We regard these as significant improvements in the formulation of, and research into YCSP as they provide a way of bridging the gap between theoretical models and their application in real-world maritime port terminals.

This paper focuses on these more practical YCSP generalizations that consider the assignment of I/O points. The problem combines the scheduling of the YC operations with the assignment and coordination of the I/O points and container time parameters (times at which containers need to be ready). The resulting problem is computationally challenging. Quite recently, Vallada et al. (2023) proposed improved mathematical models for this problem and the reported computational results indicate that a state-of-the-art solver (IBM ILOG CPLEX 20.1) is not able to prove optimality in problems as small as 10 container operators after a full hour of solver CPU time. As a result, we present heuristic approaches for the YCSP. As a scheduling problem, we build on the results of Iterated Greedy (IG) (Ruiz & Stützle, 2007), which has consistently produced state-of-the-art results for different scheduling problems. The added benefits of IG approaches result from the simplicity and ease of implementation. In this paper we present three different IG methods that incrementally introduce improvements in their operators including new destruction, reconstruction, and local search methods. We introduce a Variable Neighborhood Descent (VND) to the proposed IG methods. The algorithms are designed, calibrated, and evaluated using the Design of Experiments (DOE, Montgomery, 2017). The results are powerful, yet simple procedures that are able to yield consistently better solutions for this problem.

The rest of the paper is organized as follows: We review the existing literature of YCSPs in more details in Section 2 and formalize the researched problem in Section 3. Section 4 details the operators of the proposed IG methods. These are statistically calibrated and integrated into three IG procedures, along with comprehensive computational experiments and comparisons in Section 5. Finally, Section 6 concludes the paper and outlines future avenues of research.

2. Literature review

The yard crane scheduling problem has been analyzed, formulated and solved using different approaches depending on the characteristics of the problem. Ng and Mak (2005a, 2005b) proposed a Mixed Integer Linear Programming (MILP) model incorporating into it the yard crane routing model of Kim and Kim (1999). The YCSP was proven to be $\mathcal{NP}$-complete in Narasimhan and Palekar (2002). In all these studies, the YCSP is formulated as a machine scheduling problem with homogeneous requests, where important simplifications for the loading and movements of the yard crane are made. For example, the processing times of each container request by the crane are fixed without considering the initial and final location of the container within the yard block. Many studies make similar simplifications, for example Cao et al. (2010), Caserta et al. (2011), Chang et al. (2011), Chu et al. (2019), Dorndorf and Schneider (2010), He et al. (2010), Hsu et al. (2021), Liang et al. (2014), Yang and Jiang (2020), Zhou et al. (2020). Some studies extend these basic settings and classify container requests according to their origin and final destination. This is later detailed in Section 3. To complete requests, the YC shuttles from the original to the destination point of the container. Other authors employ Traveling Salesman Problems (TSPs) as alternative formulations (Gharehgozli et al., 2014; Vis & Roodbergen, 2009). Discussions involving different mathematical models can be found in Kemme (2011).

Among the various formulations, minimizing the YC total travel time (Gharehgozli et al., 2019, 2014; He et al., 2010; Vis & Roodbergen, 2009) or its waiting time (Li et al., 2009; Ng & Mak, 2005b; Yang & Jiang, 2020) are popular. Additionally, the completion time of all tasks, also known as makespan, is widely considered in the YCSP with multiple cranes. More recently, tardiness and/or earliness involving container release times have gained traction as a more realistic objective to consider (Chu et al., 2019; Galle et al., 2018; Vallada et al., 2023; Zheng et al., 2019).

There is also extensive research into YCSP with multiple YCs and the resulting variants. The twin or double cranes are common in most yard terminals (Gharehgozli et al., 2015). With multiple cranes, we find two main subproblems. The first one involves two or more passable cranes where each crane might deliver containers to any side (Li et al., 2012, 2009; Ng & Mak, 2005b; Vis & Carlo, 2010; Zheng et al., 2018). The second, and quite different subproblem results from a situation where two or more cranes cannot pass over each other, and the resulting interference problems that appear need to be specifically modeled, as each crane is often only capable of serving one of the sides, see Gharehgozli et al. (2015, 2017), Hu et al. (2016), Saini et al. (2017). More details about different crane systems are given in Speer and Fischer (2017), Vis and Roodbergen (2009). There is also research on how the YCSP is augmented with other important port operational problems (Hop et al., 2021; Kaveshgar & Huynh, 2015; Kizilay et al., 2020). A few representations are the YCSPs integrated with truck scheduling (Cao et al., 2010; Hsu et al., 2021), bay allocation (Lee et al., 2011), quay crane scheduling (Kaveshgar & Huynh, 2015), container relocation (Galle et al., 2018) and vehicle positioning (Zhou et al., 2020).

Scheduling problems with a single crane open up opportunities to delve into the crane loading details of real-world terminals. Some researchers have investigated the ready times (Ng & Mak, 2005a, 2005b), type of requests (Vis & Roodbergen, 2009), I/O points (Gharehgozli et al., 2014) and relocation for retrieval loading (Galle et al., 2018). As already mentioned, we regard the consideration of the I/O points as one

of the most important details introduced into YCSPs (Gharehgozli et al., 2015, 2019, 2014; Vallada et al., 2023). The I/O points, located at both extremes of the yard blocks, serve as dynamic buffers to decouple YC operations from other parts of the yard. It has to be mentioned though, that there are opportunities to augment the treatment that I/O points have received so far in the extant research. Gharehgozli et al. simplified the I/O points as they are considered infinite in quantity and have no associated costs. Vallada et al. (2023) considered the occupation of I/O points and they are modeled as resources that are dynamically occupied and released at different times. In this paper we adopt the same problem setting as Vallada et al. (2023) given how close it is to realistic settings.

As regards solution methods, we find plenty of varied approaches in the literature, from exact to heuristic, metaheuristic and hybrid procedures. Dynamic programming is employed by Vis and Roodbergen (2009) for a special YCSP using a single-row straddle carrier. Benders' decomposition is proposed in Cao et al. (2010) to tackle a particular YCSP combined with truck scheduling. Gharehgozli et al. (2019, 2014) formulated the YSCP as TSPs, and employed an ad-hoc Branch and Bound (B&B) and the Hungarian algorithms (Kuhn, 1955). Of particular interest are the effective techniques employed to eliminate the TSP subtours. A common potential limitation of all of the above approaches is scaling limitations where only small instances can be solved to optimality. For larger and practically-sized instances, heuristics and metaheuristics have been commonly proposed instead. Among these, a popular one is Genetic Algorithms (Chu et al., 2019; He et al., 2010; Hu et al., 2016; Liang et al., 2014). Other metaheuristics have been successfully applied to various YCSPs, including Simulated Annealing (Vis & Roodbergen, 2009), Tabu Search (Chen & Langevin, 2011), Particle Swarm Optimization (PSO) (Hsu et al., 2021) and Greedy Randomized Adaptive Search Procedure (GRASP) (Wang et al., 2025). There is also a rich literature on heuristics like Rolling-horizon (Chang et al., 2011; Li et al., 2012), rule-based heuristics (Zheng et al., 2019) and local search procedures (Gharehgozli et al., 2015; Vallada et al., 2023).

In conclusion, while YCSPs are well investigated, including problems with multiple cranes, we find that only Vallada et al. (2023) and Wang et al. (2025) have realistic I/O point considerations that make the problem much closer to real-life applications. This paper builds upon them, proposing solution methodologies that significantly improve the performance of the yard cranes.

3. Problem description

The YCSP entails the optimization of the operations of a single yard crane in the yard block to minimize the total weighted cost of a series of storage and retrieval requests. There are a total of n requests to schedule, each one dealing with the storage or retrieval of a container in the yard block. This block comprises R rows, B bays and T tiers (Fig. 1). We consider a block with IO^L (landside) and IO^S (seaside) I/O points in a European terminal layout. The container is either stored or retrieved by the crane and all containers need an I/O point located at the seaside or landside.

This study relies on some assumptions, that are not dissimilar to those found in the existing literature: (1) We assume a single YC in the block. (2) Containers arrive in order of the known corresponding time parameters (to be introduced later). (3) Information about reshufflings of retrieval contains and storing locations of storage containers are known in advance. (4) The YC is continuously available during operations. The consideration of time parameters for containers extends previous research where storage containers are simply assumed to have all arrived at one time or being ready at time zero. Some operational timelines in terminals are useful to define these parameters. E.g., release times of storage containers can be specified by ship berthing time and the due date by vessel departure times. Corresponding applications can be found in the literature (Chang et al., 2011; Chu et al., 2019; Galle et al., 2018; He et al., 2010; Li et al., 2012, 2009; Stahlbock & Voß, 2010; Wu et al., 2015; Zheng et al., 2018, 2019).

To solve this problem, two main decisions are required: First, determining the sequence in which the crane should handle the containers, and second, assigning an I/O point to each container. The starting position of the crane (Ini) is known, and the final position corresponds to the location of the last container in the sequence. Each container request c involves an origin location (O_c) and a destination location (D_c). The set of container requests can be categorized into four distinct groups C_1 to C_4:

- C_1 *(Sea to Yard)*: Containers arriving at the port on vessels need to be stored in the block at a specified position. Each container c has a designated arrival time, referred to as its release time (r_c). Furthermore, an I/O point on the seaside must be assigned to container c. The crane will collect the container from its assigned I/O point (O_c) and place it at its designated destination (D_c) within the block.
- C_2 *(Land to Yard)*: Containers arriving at the port from the landside by train or truck need to be stored in the block at a specified position (D_c). Each container c has also a release time, represented by r_c. In this scenario, an I/O point on the landside (O_c) must be assigned to container c.
- C_3 *(Yard to Sea)*: Containers located at a specified position (O_c) within the block need to be retrieved and loaded onto a vessel. An I/O point on the seaside (D_c) must be assigned for each container. Each container c has a due date, represented by d_c, by which it is desired to be delivered.
- C_4 *(Yard to Land)*: Containers located at a known position (O_c) within the block need to be retrieved and loaded onto a train or truck. For this, an I/O point on the landside (D_c) must be assigned. Each container c has an arrival time for the train or truck, represented as e_c, before which the container cannot be delivered. In this case, the I/O point (D_c) refers to the specific location where the train or truck has to be located to load the container.

The handling process for a container request c involves the crane starting at its initial position (Ini) and moving empty to the origin position of container c (O_c). The crane then loads c and transports it to its destination (D_c). As described in previous studies (Galle et al., 2018; Speer & Fischer, 2017), the speeds of the gantry, trolley, and hoisting mechanisms are typically standardized to 1, making the distance equivalent to time. The positions of containers and I/O points are defined by fixed coordinates (x, y, z) before delivery, corresponding to $row \geq 0$, $bay \geq 0$, and $tier \geq 0$ within the yard block. Because the crane can move vertically along rows while its trolley moves concurrently, the pairwise Chebyshev distance is often utilized in the literature for measurement (Gharehgozli et al., 2019, 2014; Vallada et al., 2023).

$$t_{i,j} = |z_i - (T+1)| + \max\{|x_i - x_j|; |y_i - y_j|\} + |(T+1) - z_j| + Res_{time} \quad (1)$$

Where Res_{time} represents the constant cost associated with relocating a container that is stacked in a middle tier and requires reshuffling before being loaded. The first and third terms in expression (1) correspond to the crane's upward and downward movements, respectively. Since the crane initiates its movement from the top, the z coordinate for the initial position is $T + 1$, making the first term of expression (1) equal to zero.

Fig. 2 illustrates the calculation of expression (1) with two adjacent containers c and c'. Fig. 2 shows empty (dashed line) and loaded moves of the YC along with time costs and locations. The YC starts an empty move from its current position Ini (initial position of the crane or previous destination of the last processed container) at time T_{now}. Then, the crane moves to the origin of container c (O_c), picks up the container and moves to its destination (D_c). The origin (O_c) and destination (D_c) of c are i and j, respectively and their coordinates are (x_i, y_i, z_i)

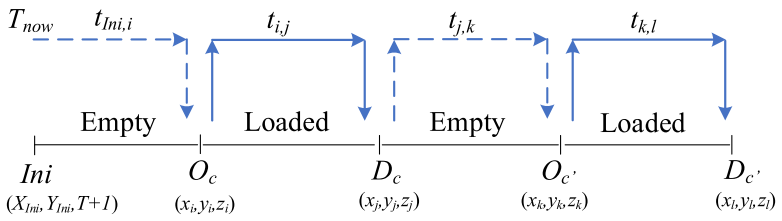

Fig. 2. Empty and loaded moves of the crane with corresponding time cost and locations for two container requests c and c'.

and (x_j, y_j, z_j). The time needed for the movements are $t_{Ini,i}$ and $t_{i,j}$, calculated according to expression (1). Once the container c is in its destination, the crane starts a new empty movement to the origin of the next container (c'). The origin ($O_{c'}$) and destination ($D_{c'}$) of container c' are k and l respectively, with coordinates (x_k, y_k, z_k) and (x_l, y_l, z_l). Times for these movements are also computed following expression (1) in the same way as for container c.

Regarding the objective function, the total weighted cost is minimized according to expression (2). Recall that two decisions are taken in this problem: First, to obtain the sequence of container requests and second the assignment of I/O points. The objective computes two type of costs, the first related to delay and the second to congestion at I/O points, penalized with weights w_i^R and w_i^C respectively. Both costs (delays and congestion) are computed in relation to the time parameter of the container (r_c for containers of type 1 and 2 and e_c for containers type 4). For containers type 3, the time parameter is the due date (d_c) of the container. In this case, if the container is retrieved before its due date, the I/O point is occupied until the due date, resulting in earliness (E_c in expression (2)) and congestion since the I/O point is not available for other requests. On the other hand, loading a type 3 container after its due date d_c results in a delay cost. Note that a container of type 3 only can generate one type of costs: delay or congestion (earliness).

$$\begin{aligned}\text{Min} \sum_{i=1,2} \sum_{c \in C_i} \left(w_i^R \left(FC_c - r_c\right) + w_i^C \left(SO_c - r_c\right)\right) \\ + \sum_{c \in C_3} \left(w_3^R \left(FO_c - d_c\right) + w_3^C E_c\right) \\ + \sum_{c \in C_4} \left(w_4^R \left(FO_c - e_c\right) + w_4^C \left(SO_c - e_c\right)\right) \end{aligned} \quad (2)$$

In expression (2), FC_c denotes the final crane time after handling container request c (the time at which the crane finishes placing container c at its destination D_c). SO_c and FO_c represent the starting and final occupation times for container c at the I/O point, respectively. These variables are computed according to Fig. 2 and expression (1) in the following way, where T_o is the time when the assigned I/O point is released:

- Storage containers $c \in C_1 \cup C_2$. Given the state of the I/O and crane, $SO_c = \max(r_c, T_o)$, and $FO_c = \max(T_{now} + t_{Ini,i}, r_c)$. Then, $FC_c = FO_c + t_{i,j}$.
- Retrieval containers $c \in C_3$. In this case, $FC_c = SO_c = \max(T_{now} + t_{Ini,i} + t_{i,j}, T_o)$, and $FO_c = \max(d_c, FC_c)$. Finally, $E_c = \max(d_c - FC_c, 0)$.
- Retrieval containers $c \in C_4$. Since the loading at the I/O point starts after e_c, $SO_c = \max(e_c, T_o)$, and $FC_c = FO_c = \max(T_{now} + t_{Ini,i} + t_{i,j}, e_c)$.

More details about the objective function, as well as a complete mathematical model, can be found in Vallada et al. (2023) and Wang et al. (2025). The YCSP model is very complex even after relaxing the I/O assignments. We have tested two simplified versions by replacing I/O assignments with a single I/O point with as much capacity as the original number of I/O points and always-free I/O points. However, even these relaxed models fail to tackle instances of more than 10 containers using mathematical programming solvers. Heuristic methods are preferable.

4. Proposed iterated greedy methods

Iterated Greedy (IG, Ruiz & Stützle, 2007), is a simple approach that has generally produced competitive results for different scheduling problems (Fanjul-Peyro & Ruiz, 2010; Hatami et al., 2015; Missaoui & Ruiz, 2022; Ruiz et al., 2019). We are not aware of any application of IG to YCSPs. We present a basic IG template, together with improvements in several of its operators to finally propose a competitive variant.

Let us first introduce a vanilla IG that will work as a baseline following the original work of Ruiz and Stützle (2007). Algorithm 1 shows the outline. In a nutshell, this basic IG, referred to as IG0 henceforth, uses the same initial solution and local search procedures of the recently proposed Greedy Randomized Adaptive Search Procedure (GRASP) by Wang et al. (2025). The solution is then destroyed and reconstructed to escape the previously found local optimum. More precisely, a number d of containers is randomly removed from the current permutation π of containers in the YC schedule in the destruction operator. These are then later reinserted back into the permutation into the best positions, one by one, during the reconstruction. Note that the local search proposed by Wang et al. (2025) is adopted to insert each container randomly into one of the top β positions among all possible options. In this procedure, I/O points are assigned using the same heuristic employed when building the initial solution. After the local search, the new solution is considered in accordance with the acceptance criterion and the procedure iterates until the stopping criterion is met.

Using the same initial solution and local search, we believe IG0 constitutes a fair comparison baseline against the GRASP of Wang et al. (2025). The destruction and reconstruction operators, as well as the acceptance criterion, are further detailed in the following sections.

Algorithm 1 : Iterated Greedy basic outline (Ruiz & Stützle, 2007).

```
1: π_0 := GenerateInitialSolution
2: π := LocalSearch(π_0)
3: π_best :=π
4: while stopping criterion is not satisfied do
5:     π_D, π_R:= Destruction (π)
6:     π' := Reconstruction (π_D, π_R)
7:     π'' := LocalSearch (π')
8:     π, π_best:= AcceptanceCriterion (π', π'', π_best)
```

4.1. Solution representation and initialization

As previously mentioned, the solution to the YCSP consists of the container sequence and I/O assignments for each container. These can be simply represented with two lists, where one is the permutation of containers and therefore, yard crane operations and the other contains the assigned I/O points for each container in the permutation. Alternatively, the presentation can also be a permutation of tuples consisting of the container and I/O point in the crane list of operators. We adopt this last representation in this paper.

IG procedures typically start from a high-quality solution. Two existing rule-based constructive methods (TPR and SETR) are tested. Among these two, TPR (Time Parameter Rule) is also the simplest, where containers are sorted in ascending order of their time parameters, r_c, d_c or e_c. TPR was proposed by Vallada et al. (2023). SETR was

recently presented by Wang et al. (2025) where containers are sorted in increasing order of the Simulated Ending Times (η_c) according to expression (3).

$$\eta_c = FC_c / w_i^R \tag{3}$$

The η_c values are estimations of the crane finishing time for container c, (FC_c) with the YC passing the median I/O points. For both methods, we have adopted the I/O assignment heuristic of Wang et al. (2025). This looks for the closest I/O points to the destination of the container c among all I/O options available from r_c, d_c, or e_c to SO_c. In the extreme case, if none of the I/O points are available, the one released first is assigned. Both constructive methods will be tested and calibrated in later sections where it will be shown that SETR produces significantly better results than TPR. For more details, readers are referred to previous papers (Vallada et al., 2023; Wang et al., 2025).

4.2. Destruction

Destruction is a fundamental operator in IG procedures. Too weak of a destruction results in the subsequent local search circling back to the previous local optimum or to a similar one. On the contrary, an overly strong destruction degenerates into a random walk. We tested two destruction methods. The first is the aforementioned IG0 destruction first proposed by Ruiz and Stützle (2007) where a number d of containers are randomly extracted from the incumbent solution. The I/O points assigned for all containers are preserved and all these removed containers will later be reinserted during the reconstruction. The second destruction operator is a novel approach that builds on the previous one adding destruction also to the I/O assignments. More precisely, after d containers are removed, d_2 randomly chosen I/O assignments of containers are destroyed by replacing their I/O points with random ones on their corresponding side. The process to select these d_2 I/O assignments is the following: We select a random I/O point and remove, at random, at most d_2 I/O assignments of containers that had this randomly chosen I/O point assigned. Note that if there are fewer than d_2 containers with this I/O point assigned, then we remove fewer I/O assignments. So basically, we are randomly removing up to d_2 I/O assignments of containers from the same randomly chosen I/O point. The rationale behind this directed destruction is that in order to deal with congestion, it is better to target specific I/O points at each destruction application. Note that the second destruction operator destroys both container sequences as well as I/O assignments. We simply refer to this second destruction operator as '$2levels$'. We also simply refer to the first operator as '$Original$'. Destruction plays a salient role in diversifying the incumbent solution after finding the local optima, which may allow for explorations of better solutions in another solution space during the following reconstruction.

4.3. Reconstruction

In the reconstruction operator we generate a complete solution, usually applying some limited form of local search, so that the new reconstructed solution is not arbitrarily bad. We also examine two reconstruction methods. The first one is again an adaptation of the baseline (referred to as '$Original$') by Ruiz and Stützle (2007). Here, each previously removed container is tested in all positions of the container sequence and finally placed into the position resulting in the best objective function value (2). Assigned I/O points are kept for all containers. Similar to the second destruction operator, we also name $2levels$ the second improved reconstruction operator. Here we also do reassignments of the I/O points after reconstructing all containers. Each reconstructed container has all potential I/O assignments on the corresponding side tested, and the one resulting in the best objective function is kept. Note that this results in a maximum of $dn + d_2 m$ calculations of the Objective Function (OF), which is significantly more expensive than the $Original$ operator. However, and as will be shown later in the experimental evaluation, this additional cost results in statistically significant improvements in the quality of the produced solutions.

A quick example illustrates the '$2levels$' destruction and reconstruction operator with five containers, from Con1 to Con5, and two I/O points, IO1 and IO2 on the same side. Fig. 3 shows the solutions before and after '$2levels$' destruction with $d = 2$ and $d_2 = 1$ where modifications are in bold. Using $d = 2$, Con1 and Con3 are removed from the solution. Employing $d_2 = 1$, we randomly select I/O point IO1 and remove $d_2 = 1$ random assignments of IO1, in this case, the I/O assignment of Con5 is destroyed and randomly replaced by assignment IO2. During the reconstruction, a random destroyed container (Con1) is chosen and inserted into the best position with the minimal objective function, e.g., position 2 in Fig. 4. Similarly, Con3 is tested in all positions and finally inserted into the last one. After having all containers reinserted into the sequence, all I/O points of the d_2 containers that had their I/O assignments removed are reconstructed. In this case, all possible I/O points are tested for Con5 and IO1 is selected.

4.4. Local search

Local search plays a crucial role in exploring and exploiting the solution space in IG methods. In this paper, we examine three different local search procedures, referred to as LS_{GR}, LS1 and VND respectively. LS_{GR} is the one presented by Wang et al. (2025) for their proposed GRASP where containers are extracted one by one and inserted into random positions from a subset of the top β best positions. LS_{GR} performed very well in the GRASP and, as mentioned, is tested here as a baseline.

Different to Wang et al. (2025), LS1, detailed in Algorithm 2, is based on the swap neighborhood. Each container swaps positions with every other container and the best pair is swapped. At each loop, LS1 reshuffles the container indexes so that the same container pairs are not examined in the same order every time. It is important to note that LS1 terminates as soon as the incumbent objective function value is improved. This results in a speedy local search. I/O assignments of containers are preserved along with the swap moves all of the time.

The third local search procedure is based on a more involved Variable Neighborhood Descent (VND) approach. The VND combines two neighborhoods. The first neighborhood deals with the container sequence and LS1 is employed. The second neighborhood explores I/O assignments, is referred to as LS_I/O, and is detailed in Algorithm 3. LS_I/O simply tests all I/O points on the corresponding side for each container in the sequence and assigns the best I/O point as a result. With these two local searches, the complete VND procedure is detailed in Algorithm 4. Note that the VND continues iterating until no improvement can be found by either LS1 or LS_I/O.

We have incorporated a problem-specific speed-up in the VND, referred to as PSK. When examining the results of LS1, we observed that most of the improvement moves correspond to pairs of containers of the same type. This is surprising as only a quarter of the possible container pairs are made up of containers of the same type for any of the calibration instances (in Section 5.1). After careful experiments on these instances, we observed that by limiting container moves only to neighbors of the same type we obtained very little (if any) degradation in solution quality and a considerable speed-up in the local search. This is expected as mixing types may result in the YC moving from one side to the other, traversing all the block. Note that to reduce any potential degradation of the solution quality, we allow a very small number of movements between different containers. More specifically, we swap all containers of the same type (25% of all containers) and 5% of the remaining 75% pairs of containers of different types, resulting in 28.75% of container pairs being examined during the local search.

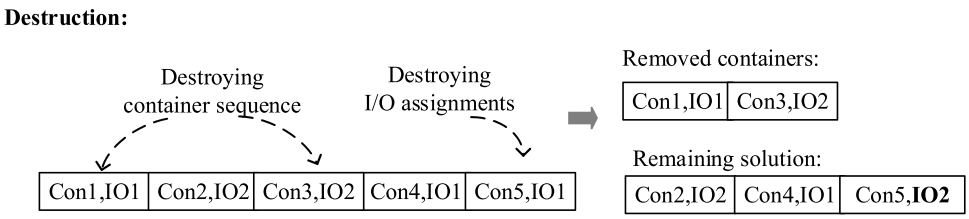

Fig. 3. Example for $2levels$ destruction operator with $d = 2$.

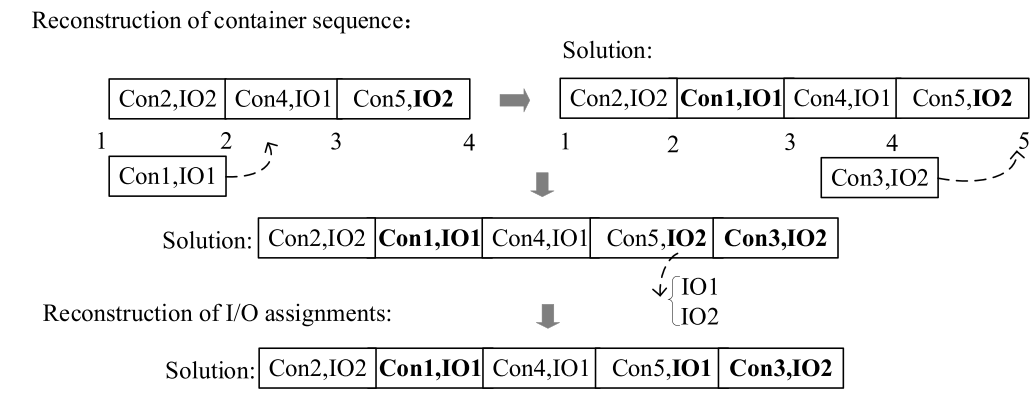

Fig. 4. Example for $2levels$ reconstruction operator.

Algorithm 2 : LS1(incumbent solution S^*, incumbent objective f^*)

```
improvement := true
while improvement := true do
    improvement = false
    C_reshf is the reshuffled index of containers in S*
    for each container i in C_reshf do      % local search on container sequence
        Exchange positions for i with all containers of the same type
        Swap i in S* with the container that obtains the best f*
        if f* is strictly improved then
            improvement := true
            break
```

Algorithm 3 : LS_I/O(incumbent solution S^*, incumbent objective f^*)

```
for each container c in S* do
    Test all other I/O options on this side for c
    O is the I/O point that obtains the best f*
    if f* is strictly improved then
        Assign I/O point O to c
```

Algorithm 4 : VND (incumbent solution S^*, incumbent objective f^*)

```
improvement := true
while improvement := true do
    improvement = false
    S' ← LS1 (S*, f*)      % local search on containers
    if f* is strictly improved then
        improvement := true
    S'' ← LS_I/O (S', f*) % local search on I/O points
    if f* is strictly improved then
        improvement := true
```

4.5. Acceptance criterion

The last step during each loop of the IG is to decide if the newly obtained solution replaces the incumbent or not. The original acceptance criterion of Ruiz and Stützle (2007) depends on a temperature parameter. In the interest of having very simple IG procedures, we adopt the parameter-less acceptance criterion of Hatami et al. (2015), where a solution is accepted if $rand \leq e^{-(RPD)}$. Where $rand$ is a uniformly distributed random number in the $[0, 1]$ range and RPD is the Relative Percentage Deviation (RPD) between the incumbent solution objective value Sol and the one from the best solution found so far ($Best$), i.e., $RPD = \frac{Sol-Best}{Best} \times 100$. The $Best$ solutions are exactly the optimal ones for all small instances. Of course, the new solution is accepted as the incumbent and the new best if it outperforms the best solution.

5. Computational and statistical experimentation

5.1. Experimental instances and settings

We carry out an extensive experimental campaign on the benchmark sets of Vallada et al. (2023) which contain a total of 720 instances. These are divided into four groups: small and small tight both with ($n \in \{5, 10\}$), medium ($n \in \{15, 20, 30, 40\}$) and large ($n \in \{50, 100, 150, 200\}$). The size of the yard is also the same used in Vallada et al. (2023) and Wang et al. (2025) where the total number of I/O points (m) is 16; 10 on the seaside and 6 on the landside in a yard block with $R = 10$ rows, $B = 42$ bays and $T = 4$ tiers. The small tight set ($n \in \{5, 10\}$) is used to study I/O congestion under the extreme case of having only one landside and two seaside I/O points. As we have a single crane and there is no interference, the crane speeds for the gantry, trolley and hoisting stages can be considered to be constant, and therefore equal to 1 without loss of generality (Speer & Fischer, 2017). The data for container requests is randomly generated. This includes container types (C_1 to C_4), yard locations (x, y, z), reshuffling time (Res_{time}, i.e., reshuffling movement) is set to 2 units, time parameters (r_c, d_c

and e_c) and weights (w_c^R and w_c^C). Time parameters follow a uniform distribution $U(0, \rho \times n \times B)$ where $\rho = \{0.1, 0.4, 0.7\}$ for storage containers and 0.4 for retrieval containers. There are two types of weights, *Equal* where all weights are identical and equal to 1, and *Non-equal* where the weights are 3 for seaside and 1 for landside containers respectively. For calibration purposes, a smaller data set with 144 instances was generated in Wang et al. (2025) with 2 replicates of 72 parameter combinations. We also employ this test to calibrate the proposed IG algorithms.

All IG methods and the compared GRASP are coded in C# and compiled with Microsoft Visual Studio 2019. We tested all instances in virtual x64 Windows 10 OS machines, each one configured with 16 GBytes RAM memory and four virtual cores. These virtual machines are run on the OpenStack virtualization platform powered by several servers, each one equipped with two 18 core Intel Xeon Gold 5220 processors running at 2.2 GHz and 384 GBytes of RAM. All computational tasks are randomly distributed across these servers without any parallel coding or computing.

5.2. Calibration experiments and tested methods

Calibration of IG operators is carried out using the Design of Experiments (DOE) approach together with the Analysis of Variance (ANOVA) statistical technique (Montgomery, 2017). ANOVA is a statistical model used to study if different factors have a significant effect on a response variable. It has to be noted that ANOVA is a parametric technique and usually three hypotheses about the data need to be met. These are normality, homoscedasticity and the independence of residuals. While ANOVA is very robust w.r.t. to mild to severe violations of normality and homoscedasticity, special care needs to be put into the independence of the residuals. However, in computer experimentation, this is largely the case as one treatment, entailing an algorithm configuration runs and terminates and there is no effect on the next algorithm run. Randomization of experiments also helps with independence.

We start with a complete factorial DOE with 8 factors (all combinations of the factor's levels and variants are studied). The results are analyzed in a multi-factor ANOVA. The 8 studied factors potentially affect the performance of the proposed IG methods. Instance factors n and m are uncontrollable as they are given by the instance sizes but are included in the experiment so as to be able to study how the different IG factors interact with instance sizes. Each instance is run several times with different random seeds to get replicas in the experiment. The different replica numbers are used as a witness factor, which is expected to be insignificant, i.e., the replica number should not have a statistically significant effect. The five remaining factors are controllable and are: (1) The heuristic used for obtaining the initial solution, referred to as IniMethod and set at two variants: TPR and SETR; (2) The number of destroyed containers (d) in the destruction operator, tested at 5 levels: {3, 6, 9, 12, 15}; (3) The destruction size of I/O points (d_2), tested at 4 levels: {0, 2, 4, 6}. Note how level 0 is a witness level as it implies not destroying I/O points at all; (4) Local search methods (LSType) at three variants: LS_{GR}, LS1, and VND; (5) The local search speed-up (PSK) at two variants: 1 (enabled) and 0 (disabled). The RPD is the response variable in the ANOVA. In total, there are $2 \times 5 \times 4 \times 3 \times 2 = 240$ combinations for the controlled factors. Each calibration instance is tested for 10 replicates using the replica number as a random seed as a variance reduction technique (McGeoch, 1992). We employ the same stopping CPU time of Wang et al. (2025), as $n \times \ln n \times m \times \tau$ milliseconds, where $\tau = 10$ for the 144 calibration instances. As a result, we have run $240 \times 144 \times 10 = 345,600$ experiments with a total CPU time usage of 3,652 h, or about 152 days.

The detailed ANOVA tables from the experiment are omitted for the sake of brevity. Instead, we provide them as an online appendix with complete supporting data for the calibrations. Among the five controllable factors, LSType is the most significant, resulting in a very large F-ratio. This is illustrated in Fig. 5 by wide level gaps and very narrow (almost non-existent) confidence intervals around the means. It is shown that VND clearly outperforms the other two local search methods. Overly large F-ratios tend to affect the normality and homoscedasticity hypotheses (Montgomery, 2017). Basically, statistical testing is not needed when the results are very evident. Therefore, LSType is fixed at the corresponding level for each algorithm in the calibrations of the different IG procedures. This results in three IG variants: IG1, IG2 and IG3, with key modifications from the baseline IG0. Table 1 presents the basic settings for all IG methods.

Table 1
Configurations for the proposed IG algorithms.

Algorithm	Destruction	Local search	PSK
IG0	*Original*	LS_{GR}	No
IG1	*Original*	LS1	No
IG2	$2levels$	VND	No
IG3	$2levels$	VND	Yes

The IG methods are calibrated with the remaining controllable factors. IG0 and IG1 have only two factors, IniMethod and d, that need calibration. IniMethod is more significant than d for IG0 and IG1 with SETR being the better constructive heuristic. As a matter of fact, SETR outperforms TPR for all the proposed IG procedures. The means plots for IG0 and IG1 are given in the online materials, where 9 is the best destruction size. Fig. 6 shows the means plots for IG2 with subfigures to the right in decreasing order of statistical significance. For IG2, the number of containers to be removed is set at 12. As can be observed, d_2 at levels 2–6 is better than at level 0, meaning that the proposed $2levels$ destruction operator with I/O destruction is better than *Original*. We set DesIO at 2 as it is slightly better than 4, although 2 and 4 are statistically equivalent at 95% HSD confidence across all instances (although one could zoom in for specific instance size intersections, we prefer not to over-calibrate). Performing a similar analysis for IG3 results in similar means plots given in Fig. 7. Here d_2=2 is statically better than values 4 and 6 and much better than 0, showing again that destroying a minimal number of I/O points is best. Of special interest is the factor PSK. The means plot in Fig. 6 indicates that the local search speed-up provides some small, but statistically significant performance gains.

We performed additional experiments to empirically prove that the proposed IG methods are quite robust. In this experiment, the best IG proposal was tested using random levels for the factors considered in the calibration. The details are given in the online materials. The results indicate that the best IG proposal with random parameter values is sound and still better than the state-of-the-art approaches.

5.3. Comparison of methods

We now compare the proposed IG algorithms against the 3-Local Search Method (LSM) of Vallada et al. (2023) and the GRASP (GR) of Wang et al. (2025). Each algorithm is independently run five times for all 720 test instances. Tables 2 and 3 show average results grouped by container size n where each cell is the average of 60 instances and 5 runs (300 results). Note that the results of LSM consist of the best solutions among 3 algorithms of Vallada et al. (2023), TPR_{ls}, $MRTP_{ls}$ and NCR_{ls}. Small instances are run with $\tau = 10$ only as more time is not needed. Medium and large instances are run three independent times with $\tau = 10$, 20 and 30. It has to be stressed that LSM uses n seconds as a stopping criterion, which is always higher than the stopping criterion employed for the other algorithms. This is still true even for $\tau = 30$.

Regarding the small instances results in Table 2, we see that IG0 has found 211 optimums out of 240 instances, far more than LSM and GR. Note that LSM is at the same time slower. The number of optimums increases with IG1, IG2 and IG3 to 229 (70.9% higher) with a 97.3% smaller $ARPD$ of 0.04%. We can observe that the proposed IG methods

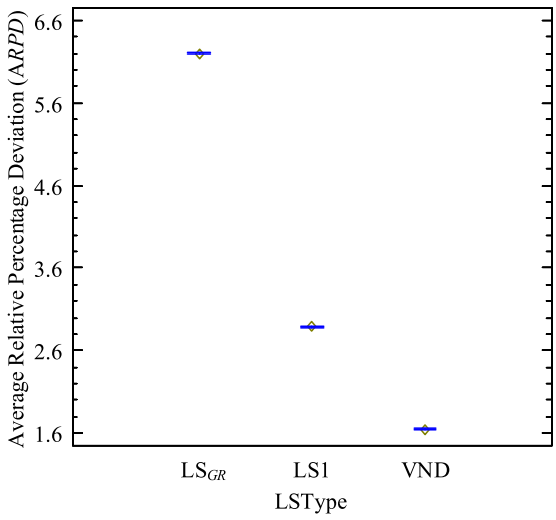

Fig. 5. Means plots for the tested local search procedures factor (LSType) in the ANOVA calibration with 95% Tukey's Honest Significant Difference (HSD) confidence intervals.

Table 2
Number of optimums (#Opt.) and Average Relative Percentage Deviation (*ARPD*) for small and small tight (ST) instances. CPU time for CPLEX (MP) is set to 3600 s (time for LSM in parentheses), while all other methods share CPU times given in the last column ($\tau = 10$). Best results in bold.

	n	#	MP	LSM (s)	GR	IG0	IG1	IG2	IG3	Time (s.)
#Opt.	5(ST)	60	60	43 (15)	54	**58**	**58**	**58**	**58**	**0.24**
	10(ST)	60	56	19 (30)	37	**57**	**57**	**57**	**57**	**0.69**
	5	60	60	52 (15)	54	**58**	**58**	**58**	**58**	**1.29**
	10	60	45	20 (30)	26	38	**56**	**56**	**56**	**3.68**
Total		240	221	134 (90)	171	211	**229**	**229**	**229**	**5.90**
ARPD	5(ST)	60	0	0.96 (15)	0.45	0.07	**0.06**	**0.06**	**0.06**	**0.24**
	10(ST)	60	2.61	3.24 (30)	0.88	0.05	**0.04**	**0.04**	**0.04**	**0.69**
	5	60	0	0.50 (15)	0.19	**0.04**	**0.04**	**0.04**	**0.04**	**1.29**
	10	60	3.60	1.15 (30)	0.74	0.19	**0.02**	**0.02**	**0.02**	**3.68**
Average		60	1.56	1.46 (22.5)	0.56	0.09	**0.04**	**0.04**	**0.04**	**1.48**

can solve to optimality most small instances within a few seconds, much efficient than the LSM and CPLEX results. Vallada et al. (2023).

As for the medium and large instances results in Table 3, IG0 has produced 685 new best solutions for the 720 instances from the previous results of the GRASP of Wang et al. (2025) (GR). IG0 improves the *ARPD* by 64.6% and by 35.7% from LSM and GR for the medium and large instances and across all τ values respectively. We stress that IG0 is a very simple IG method using the same initial solution and local search as GR.

Among all proposed IG methods, IG3 performs the best. Let us start with IG1. IG1 outperforms IG0 for all instances. The overall *ARPD* of IG1 is 67.8% lower than IG0, widening the gap with LSM and GR. IG2 and IG3 go much further, with very small *ARPD* values. We attribute this to the perturbation on I/O assignments from new IG operators like $2levels$ destruction and the VND local search. On average, IG2 produces 81.7% smaller *ARPD* than IG1. Clearly, IG3 is the best proposed alternative. As a matter of fact, it completely outperformed the best GRASP of Wang et al. (2025) (GR), producing better results in all 720 instances. In general, IG3 results in 98.2% and 96.8% lower *ARPD* values when compared to LSM and GR respectively.

While we expect almost all of these differences to be statistically significant, we do carry out statistical testing to formally assess this out-performance. This time, a non-parametric statistical test, Wilcoxon signed-rank (García et al., 2009), is employed. We compare the *ARPD* results of all 720 instances. The number of instances that IG3 performs better, worse or equal to other methods is labeled with '-', '+' and '=',

Table 3
Average Relative Percentage Deviation (*ARPD*) and time of medium and large instances for the compared methods grouped by CPU time (τ) and number of containers (n) and instances (#). CPU times (in seconds) for LSM are given in parenthesis whereas for all other methods, which share CPU times, they are given in the last column. Best results in bold.

τ	n	#	LSM (s)	GR	IG0	IG1	IG2	IG3	Time (s)
10	15	60	4.01 (45)	1.20	0.59	0.02	**0.00**	**0.00**	**6.50**
	20	60	6.82 (60)	2.74	1.67	0.15	0.01	**0.00**	**9.59**
	30	60	11.61 (90)	5.87	3.83	0.81	0.12	**0.11**	**16.33**
	40	60	15.82 (120)	8.84	5.60	1.63	**0.24**	0.29	**23.61**
	50	60	16.90 (150)	11.08	6.52	2.35	**0.41**	0.53	**31.30**
	100	60	21.51 (300)	14.84	9.32	3.83	0.95	**0.86**	**73.68**
	150	60	25.45 (450)	15.65	11.03	4.18	1.14	**0.88**	**120.26**
	200	60	32.85 (600)	16.07	12.09	4.03	1.36	**1.14**	**169.55**
Average		60	16.87 (226)	9.54	6.33	2.12	0.53	**0.48**	**56.35**
20	15	60	4.01 (45)	1.20	0.48	0.01	**0.00**	**0.00**	**13.00**
	20	60	6.82 (60)	2.66	1.59	0.10	0.01	**0.00**	**19.17**
	30	60	11.61 (90)	5.59	3.73	0.60	0.11	**0.08**	**32.65**
	40	60	15.82 (120)	8.55	5.27	1.44	**0.14**	0.23	**47.22**
	50	60	16.90 (150)	10.52	6.21	2.11	**0.26**	0.34	**62.59**
	100	60	21.51 (300)	14.35	8.37	3.45	0.67	**0.41**	**147.37**
	150	60	25.45 (450)	15.28	10.29	3.63	0.64	**0.48**	**240.51**
	200	60	32.85 (600)	15.71	11.24	3.51	0.68	**0.53**	**339.09**
Average		60	16.87 (226)	9.23	5.90	1.86	0.31	**0.26**	**112.70**
30	15	60	4.01 (45)	1.20	0.44	0.01	**0.00**	**0.00**	**19.50**
	20	60	6.82 (60)	2.54	1.46	0.14	0.01	**0.00**	**28.76**
	30	60	11.61 (90)	5.52	3.65	0.47	0.10	**0.06**	**48.98**
	40	60	15.82 (120)	8.25	5.15	1.16	**0.11**	0.20	**70.83**
	50	60	16.90 (150)	10.31	6.04	2.02	**0.21**	0.28	**93.89**
	100	60	21.51 (300)	14.12	8.03	3.08	0.42	**0.30**	**221.05**
	150	60	25.45 (450)	15.07	9.94	3.71	0.41	**0.31**	**360.77**
	200	60	32.85 (600)	15.50	10.85	3.65	0.38	**0.31**	**508.64**
Average		60	16.87 (226)	9.06	5.69	1.78	0.20	**0.18**	**169.05**
Tot. Ave.		60	16.87 (226)	9.28	5.97	1.92	0.35	**0.31**	**112.70**

Table 4
Significance of the Wilcoxon's signed-rank test between IG3 and the other algorithms at $\alpha = 0.05$ significance level for Average Relative Percentage Deviation (*ARPD*) across 720 instances for each τ stopping time.

τ	Method	-/+/=	R+	R-	p-value
10	LSM	580/1/139	1	169 070	0.000
	GR	546/0/174	0	149 331	0.000
	IG0	487/1/232	3	119 313	0.000
	IG1	381/6/333	640	74 438	0.000
	IG2	199/130/391	22 524	31 761	0.007
20	LSM	580/1/139	1	169 070	0.000
	GR	546/0/174	0	149 331	0.000
	IG0	478/2/240	52	115 388	0.000
	IG1	368/6/346	691	69 434	0.000
	IG2	188/134/398	22 674	29 329	0.047
30	LSM	580/1/139	1	169 070	0.000
	GR	546/0/174	0	149 331	0.000
	IG0	473/2/245	48	113 002	0.000
	IG1	354/5/361	287	64 333	0.000
	IG2	172/140/408	21 905	28 498	0.044

respectively, in Table 4, where R- represents the sum of negative/better ranks and R+, positive/worse ranks. Recall that the values of R+ and R-, to a certain extent, reflect the degree of difference between IG3 and the compared method. With a 95% confidence level, IG3 offers statistically superior results to LSM, GR, IG0, IG1 and IG2 for all three CPU time stopping criteria.

We now study the behavior and effect of CPU time stopping criteria and instance sizes on the proposed IG algorithms. A new full factorial design studying the proposed methods, n and τ as factors along with *RPD* as a response variable is carried out. We focus on the high performing methods and disregard LSM. Fig. 8 plots the interactions with 95% Turkey HSD intervals between the compared methods and container sizes, as well as CPU times. As can be observed, with larger

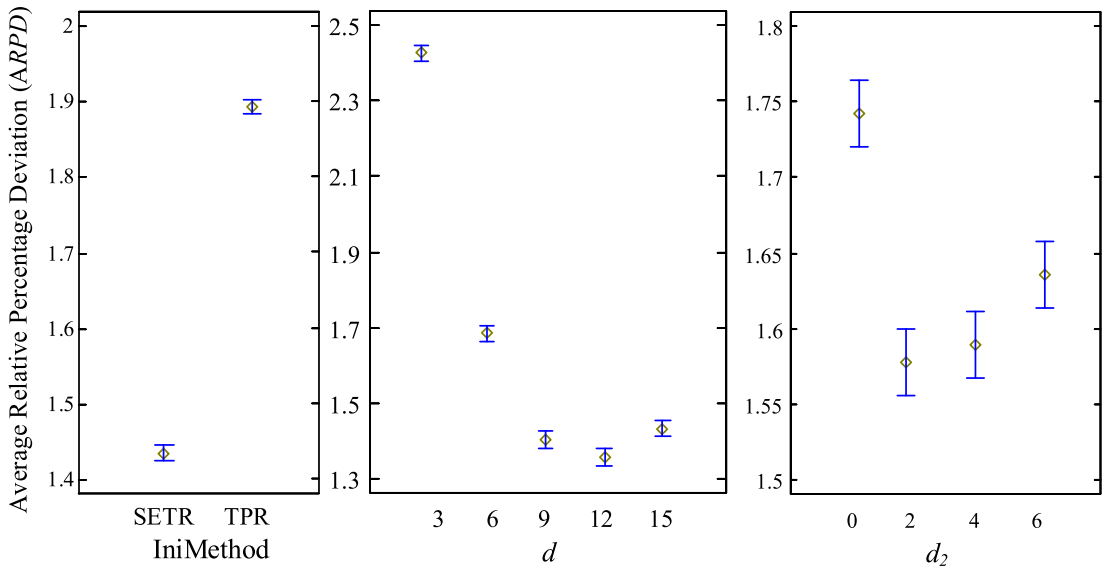

Fig. 6. Means plots for all the factors in the ANOVA calibration for IG2 with 95% Tukey's Honest Significant Difference (HSD) confidence intervals.

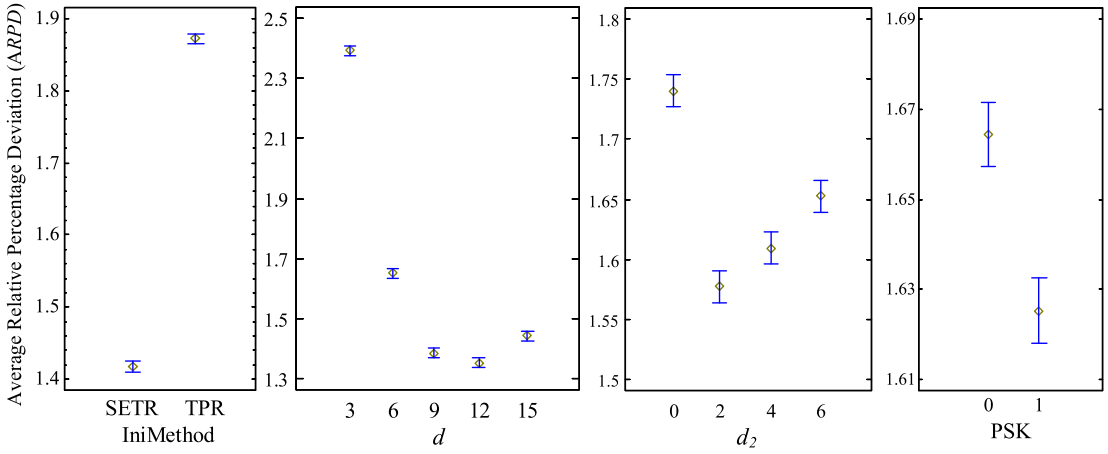

Fig. 7. Means plots for all the factors in the ANOVA calibration for IG3 with 95% Tukey's Honest Significant Difference (HSD) confidence intervals.

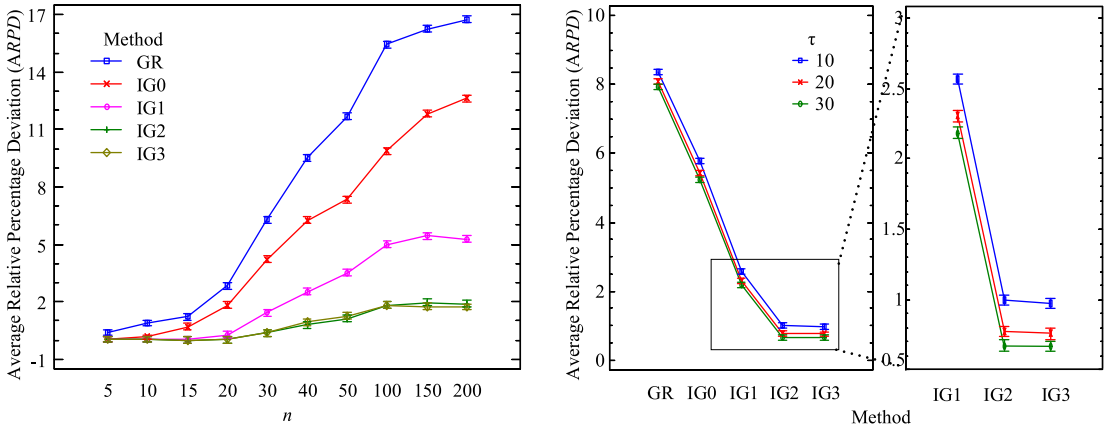

Fig. 8. Means plot of the interaction between container size and methods (left) and methods and CPU times (right) with 95% Tukey's Honest Significant Difference (HSD) confidence intervals.

instances, the gap between IG2/IG3 and GR and IG0 widens. This supports the hypothesis that the VND local search is very effective even for large problems. Results between $\tau = 20$ and 30 do not differ much, indicating convergence.

A very similar ANOVA examines the IG methods under deviations of time parameters while adopting ρ and methods as factors. The time-varying parameter ρ governs different time parameter generations introduced in Section 5.1. Fig. 9 presents the means of the time-varying

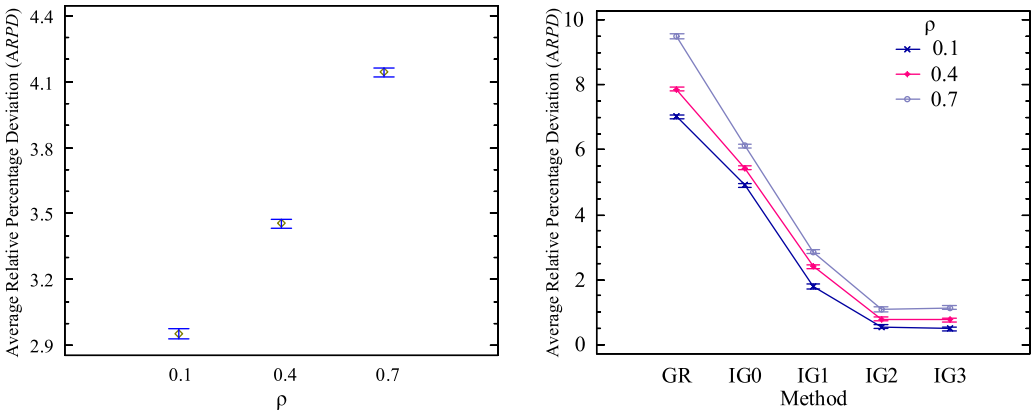

Fig. 9. Means plot of the Time-Varying Parameter ρ (left) and the interaction between ρ and methods (right) with 95% Tukey's Honest Significant Difference (HSD) confidence intervals.

parameter and its interaction with methods with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. As we expected, the time-varying parameter exhibits strong significance for different levels. Meanwhile, it confirms that the proposed IG methods are quite robust, outperforming the GR algorithm across various time parameter levels.

6. Conclusions and future research

Scheduling problems on terminal yards are complex in nature and challenging to optimize. In this paper we have considered a realistic yard crane scheduling problem where the yard crane transports containers in cooperation with subsystems like trucks, vessels and I/O points. Decisions are made for container loading sequences as well as for I/O point assignments. We have proposed four Iterated Greedy (IG) methods with an increasing number of improvements in the IG operators. Simple, yet very effective enhancements in the destruction and reconstruction operators, working both with container sequences and I/O assignments, have been presented. Two effective local search methods have been proposed, including a Variable Neighborhood Descent (VND). All IG methods are very simple, easy to code, calibrate and extend. After calibrations and comparisons with the existing literature, we have concluded that the simplest proposed IG0 significantly and statistically outperforms the best of the three methods (LSM) presented in Vallada et al. (2023) and the recent GRASP by Wang et al. (2025) by 64.6% and 35.7% on the Average Relative Percentage Deviation ($ARPD$) metric for all instances respectively. The proposed IG methods with additional improved operators are still very simple but outperform IG0 by a significant margin. These improvements come from simple diversification and intensification mechanisms over IG0 and not as a result of complex problem-specific knowledge. The best proposed method, IG3, outperforms LSM by 98.2% and GRASP (GR) by 96.8%.

Solving realistic optimization problems, stemming from terminal operations and improving operational efficiency at ports, is of paramount importance and yard crane scheduling problems play a significant role in this. Future research may entail the extension of research to multiple yard cranes or other mixed problems e.g., container reshuffling and uncertainty of time parameters. Furthermore, there is an opportunity to incorporate the dynamics of the problem where storage containers need empty yard positions, and retrieval containers generate them, as a result, incorporating position recycling is promising research.

CRediT authorship contribution statement

Hongtao Wang: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Rubén Ruiz:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Fulgencia Villa:** Writing – review & editing, Supervision, Funding acquisition. **Eva Vallada:** Writing – review & editing, Supervision, Funding acquisition.

Acknowledgments

The authors are partially sponsored by the Spanish Ministry of Science and Innovation under the project "OPRES-Realistic Optimization in Problems in Public Health"(No. PID2021-124975OB-I00) partially financed with FEDER funds.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ejor.2025.04.052.

References

Cao, J. X., Lee, D. H., Chen, J. H., & Shi, Q. (2010). The integrated yard truck and yard crane scheduling problem: Benders' decomposition-based methods. *Transportation Research Part E: Logistics and Transportation Review*, *46*(3), 344–353. http://dx.doi.org/10.1016/j.tre.2009.08.012.

Caserta, M., Schwarze, S., & Voß, S. (2011). Container rehandling at maritime container terminals. In *Handbook of terminal planning*: *Vol. 49*, (pp. 247–269). Springer Science & Business Media, http://dx.doi.org/10.1007/978-3-030-39990-0_16.

Chang, D., Jiang, Z., Yan, W., & He, J. (2011). Developing a dynamic rolling-horizon decision strategy for yard crane scheduling. *Advanced Engineering Informatics*, *25*(3), 485–494. http://dx.doi.org/10.1016/j.aei.2011.02.003.

Chen, L., & Langevin, A. (2011). Multiple yard cranes scheduling for loading operations in a container terminal. *Engineering Optimization*, *43*(11), 1205–1221. http://dx.doi.org/10.1080/0305215X.2010.548865.

Chu, F., He, J., Zheng, F., & Liu, M. (2019). Scheduling multiple yard cranes in two adjacent container blocks with position-dependent processing times. *Computers & Industrial Engineering*, *136*, 355–365. http://dx.doi.org/10.1016/j.cie.2019.07.013.

Dorndorf, U., & Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum*, *32*(3), 617–632. http://dx.doi.org/10.1007/s00291-010-0206-3.

Fanjul-Peyro, L., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, *207*(1), 55–69. http://dx.doi.org/10.1016/j.ejor.2010.03.030.

Galle, V., Barnhart, C., & Jaillet, P. (2018). Yard crane scheduling for container storage, retrieval, and relocation. *European Journal of Operational Research*, *271*(1), 288–316. http://dx.doi.org/10.1016/j.ejor.2018.05.007.

García, S., Molina, D., Lozano, M., & Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. *Journal of Heuristics*, *15*, 617–644. http://dx.doi.org/10.1007/s10732-008-9080-4.

Gharehgozli, A. H., Laporte, G., Yu, Y., & de Koster, R. (2015). Scheduling twin yard cranes in a container block. *Transportation Science*, *49*(3), 686–705. http://dx.doi.org/10.1287/trsc.2014.0533.

Gharehgozli, A. H., Vernooij, F. G., & Zaerpour, N. (2017). A simulation study of the performance of twin automated stacking cranes at a seaport container terminal. *European Journal of Operational Research*, *261*(1), 108–128. http://dx.doi.org/10.1016/j.ejor.2017.01.037.

Gharehgozli, A., Yu, Y., de Koster, R., & Du, S. (2019). Sequencing storage and retrieval requests in a container block with multiple open locations. *Transportation Research Part E: Logistics and Transportation Review*, *125*(5), 261–284. http://dx.doi.org/10.1016/j.tre.2019.03.008.

Gharehgozli, A. H., Yu, Y., de Koster, R., & Udding, J. T. (2014). An exact method for scheduling a yard crane. *European Journal of Operational Research*, *235*(2), 431–447. http://dx.doi.org/10.1016/j.ejor.2013.09.038.

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, *169*(11), 76–88. http://dx.doi.org/10.1016/j.ijpe.2015.07.027.

He, J., Chang, D., Mi, W., & Yan, W. (2010). A hybrid parallel genetic algorithm for yard crane scheduling. *Transportation Research Part E: Logistics and Transportation Review*, *46*(1), 136–155. http://dx.doi.org/10.1016/j.tre.2009.07.002.

Hop, D. C., Van Hop, N., & Anh, T. T. M. (2021). Adaptive particle swarm optimization for integrated quay crane and yard truck scheduling problem. *Computers & Industrial Engineering*, *153*, Article 107075. http://dx.doi.org/10.1016/j.cie.2020.107075.

Hsu, H. P., Tai, H. H., Wang, C. N., & Chou, C. C. (2021). Scheduling of collaborative operations of yard cranes and yard trucks for export containers using hybrid approaches. *Advanced Engineering Informatics*, *48*, Article 101292. http://dx.doi.org/10.1016/j.aei.2021.101292.

Hu, Z. H., Sheu, J. B., & Luo, J. X. (2016). Sequencing twin automated stacking cranes in a block at automated container terminal. *Transportation Research Part C: Emerging Technologies*, *69*(8), 208–227. http://dx.doi.org/10.1016/j.trc.2016.06.004.

Kaveshgar, N., & Huynh, N. (2015). Integrated quay crane and yard truck scheduling for unloading inbound containers. *International Journal of Production Economics*, *159*(1), 168–177. http://dx.doi.org/10.1016/j.ijpe.2014.09.028.

Kemme, N. (2011). RMG crane scheduling and stacking: overview and implications on terminal planning. In *Handbook of terminal planning* (pp. 271–301). Springer.

Kim, K. H., & Kim, K. Y. (1999). An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science*, *33*(1), 17–33. http://dx.doi.org/10.1287/trsc.33.1.17.

Kizilay, D., Van Hentenryck, P., & Eliiyi, D. T. (2020). Constraint programming models for integrated container terminal operations. *European Journal of Operational Research*, *286*(3), 945–962. http://dx.doi.org/10.1016/j.ejor.2020.04.025.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*(1–2), 83–97. http://dx.doi.org/10.1002/nav.3800020109.

Lee, D. H., Jin, J. G., & Chen, J. H. (2011). Integrated bay allocation and yard crane scheduling problem for transshipment containers. *Transportation Research Record: Journal of the Transportation Research Board*, *2222*(1), 63–71. http://dx.doi.org/10.3141/2222-08.

Li, W., Goh, M., Wu, Y., Petering, M., de Souza, R., & Wu, Y. (2012). A continuous time model for multiple yard crane scheduling with last minute job arrivals. *International Journal of Production Economics*, *136*(2), 332–343. http://dx.doi.org/10.1007/978-3-319-17025-1_4.

Li, W., Wu, Y., Petering, M., Goh, M., & Souza, R. d. (2009). Discrete time model and algorithms for container yard crane scheduling. *European Journal of Operational Research*, *198*(1), 165–172. http://dx.doi.org/10.1007/978-3-319-17025-1_3.

Liang, C. J., Chen, M., Gen, M., & Jo, J. (2014). A multi-objective genetic algorithm for yard crane scheduling problem with multiple work lines. *Journal of Intelligent Manufacturing*, *25*(5), 1013–1024. http://dx.doi.org/10.1007/s10845-013-0792-4.

McGeoch, C. (1992). Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. *ACM Computing Surveys*, *24*(2), 195–212. http://dx.doi.org/10.1016/B978-0-12-415825-2.00010-3.

Missaoui, A., & Ruiz, R. (2022). A parameter-less iterated greedy method for the hybrid flowshop scheduling problem with setup times and due date windows. *European Journal of Operational Research*, *303*(1), 99–113. http://dx.doi.org/10.1016/j.ejor.2022.02.019.

Montgomery, D. C. (2017). *Design and analysis of experiments* (8th ed.). John Wiley & Sons.

Narasimhan, A., & Palekar, U. S. (2002). Analysis and algorithms for the transtainer routing problem in container port operations. *Transportation Science*, *36*(1), 63–78. http://dx.doi.org/10.1287/trsc.36.1.63.576.

Ng, W. C., & Mak, K. L. (2005a). An effective heuristic for scheduling a yard crane to handle jobs with different ready times. *Engineering Optimization*, *37*(8), 867–877. http://dx.doi.org/10.1080/03052150500323849.

Ng, W. C., & Mak, K. L. (2005b). Yard crane scheduling in port container terminals. *Applied Mathematical Modelling*, *29*(3), 263–276. http://dx.doi.org/10.1016/j.apm.2004.09.009.

Ruiz, R., Pan, Q. K., & Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, *83*(3), 213–222. http://dx.doi.org/10.1016/j.omega.2018.03.004.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Rsearch*, *177*(3), 2033–2049. http://dx.doi.org/10.1016/j.ejor.2005.12.009.

Saini, S., Roy, D., & de Koster, R. (2017). A stochastic model for the throughput analysis of passing dual yard cranes. *Computers & Operations Research*, *87*(11), 40–51. http://dx.doi.org/10.1016/j.cor.2017.05.012.

Speer, U., & Fischer, K. (2017). Scheduling of different automated yard crane systems at container terminals. *Transportation Science*, *51*(1), 305–324. http://dx.doi.org/10.1287/trsc.2016.0687.

Stahlbock, R., & Voß, S. (2010). Efficiency considerations for sequencing and scheduling of double-rail-mounted gantry cranes at maritime container terminals. *International Journal of Shipping and Transport Logistics*, *2*(1), 95–123. http://dx.doi.org/10.1504/IJSTL.2010.029899.

UNCTAD (2023). Review of maritime transport 2022. Navigating stormy waters. In *United Nations conference on trade and development*: *Technical report*, United Nations Publications, URL: https://unctad.org/system/files/official-document/rmt2023_en.pdf.

Vallada, E., Belenguer, J. M., Villa, F., & Alvarez-Valdes, R. (2023). Models and algorithms for a yard crane scheduling problem in container ports. *European Journal of Operational Research*, *309*(2), 910–924. http://dx.doi.org/10.1016/j.ejor.2023.01.047.

Vis, I. F., & Carlo, H. J. (2010). Sequencing two cooperating automated stacking cranes in a container terminal. *Transportation Science*, *44*(2), 169–182. http://dx.doi.org/10.1287/trsc.1090.0298.

Vis, I. F., & Roodbergen, K. J. (2009). Scheduling of container storage and retrieval. *Operations Research*, *57*(2), 456–467. http://dx.doi.org/10.1287/opre.1080.0621.

Wang, H., Villa, F., Vallada, E., & Ruiz, R. (2025). Solving the yard crane scheduling problem with dynamic assignment of input/output points. *Computers & Operations Research*, *173*, Article 106853. http://dx.doi.org/10.1016/j.cor.2024.106853.

Wu, Y., Li, W., Petering, M. E. H., Goh, M., & Souza, R. d. (2015). Scheduling multiple yard cranes with crane interference and safety distance requirement. *Transportation Science*, *49*(4), 990–1005. http://dx.doi.org/10.1287/trsc.2015.0641.

Yang, X. M., & Jiang, X. J. (2020). Yard crane scheduling in the ground trolley-based automated container terminal. *Asia-Pacific Journal of Operational Research*, *37*(2), Article 2050007. http://dx.doi.org/10.1142/S0217595920500074.

Zheng, F., Man, X., Chu, F., Liu, M., & Chu, C. (2018). Two yard crane scheduling with dynamic processing time and interference. *IEEE Transactions on Intelligent Transportation Systems*, *19*(12), 3775–3784. http://dx.doi.org/10.1109/TITS.2017.2780256.

Zheng, F., Man, X., Chu, F., Liu, M., & Chu, C. (2019). A two-stage stochastic programming for single yard crane scheduling with uncertain release times of retrieval tasks. *International Journal of Production Research*, *57*(13), 4132–4147. http://dx.doi.org/10.1080/00207543.2018.1516903.

Zhou, C., Lee, B. K., & Li, H. (2020). Integrated optimization on yard crane scheduling and vehicle positioning at container yards. *Transportation Research Part E: Logistics and Transportation Review*, *138*(6), Article 101966. http://dx.doi.org/10.1016/j.tre.2020.101966.