Computers & Industrial Engineering 189 (2024) 109978

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Solving a short sea inventory routing problem in the oil industry

Sergio Cavero [a], Manuel Laguna [b], Eduardo G. Pardo [a,*]

[a] Universidad Rey Juan Carlos, Spain
[b] University of Colorado Boulder, United States

ARTICLE INFO

Keywords:
Inventory routing
Multiperiod sea transportation
Heuristics

ABSTRACT

We address an optimization problem related to the minimization of the distribution costs associated with product delivery in the oil industry. Particularly, the problem consists of determining a schedule of shipments from production ports to satisfy demand and desired inventory limits at consumptions ports. Products are transported in vessels, which can be viewed as a set of shared resources. The complexity of the problem derives from the problem structure and the number of decisions that need to be made throughout a planning horizon. The context that we studied belongs to the family of short sea inventory routing problems for which the ports are in the same geographical area. We formulate a mixed-integer programming model that captures the most relevant features of the real system. The main decisions include the selection of the vessels that will be used, the paths that each vessel will follow, and the quantities of each product loaded and unloaded at each port visited during the planning horizon. We test the limits of our mathematical programming formulation and develop a heuristic approach for tackling problem sizes that exceed the capabilities of a commercial solver.

1. Introduction

Oil companies, like many vertically integrated organizations, move bulk and semi-bulk products among their facilities via waterways. Specifically, they ship crude oil from oil fields to refineries and ship refined products from refineries to storage facilities (i.e., tank farms that act as distribution centers). The relevant decisions in this setting consist of determining the amounts of each product to ship from each origin to each destination, when to deliver these amounts, and what route to follow to make these deliveries. This situation falls within the decision-making problems known in the literature as inventory routing. The main difference between inventory routing and vehicle routing problems, which have been extensively studied in the literature, is that in vehicle routing the amounts to be delivered are known and therefore the focus is on finding a set of routes to make the deliveries at a minimum cost. In inventory routing, on the other hand, quantities must be determined to satisfy demand while keeping stock within desired levels.

As pointed out by Ronen (2002), there are also some major differences between maritime inventory routing problems and inventory routing in trucks. For instance, vessels, as opposed to trucks, are usually different from each other with respect to capacity, compartmentalization, and port accessibility. In addition, products (such as fuel) must be carried in different compartments of the vessel as opposed to packaged products that can be stored in the same compartment of a truck.

We engaged in a project with a major oil producer to create a scheduling tool for its inventory routing problem in the Gulf of Mexico. The ports in this study are on the coast of Texas, Louisiana, and Florida. Because the ports are in a single geographical area, the problem belongs to the family known as short sea inventory routing problems (SSIRP), also referred to in the literature as the inventory constrained maritime routing problem (ICMRP) (Stanzani et al., 2018; Diz, Oliveira, & Hamacher, 2017). In contrast, deep sea inventory routing involves transportation across continents. The literature in the last twenty years reflects the general interest in these maritime inventory routing problems (MIRP). Comprehensive reviews of this literature can be found in (Christiansen, Fagerholt, & Ronen, 2004, 2007, 2013).

Although MIRPs have several common elements, they also include features and characteristics that make each situation unique. The general approach that researchers and practitioners have followed is to model these problems as mathematical programs to determine the boundary of what is solvable within practical time limits. Advances in commercial mathematical programming software and the availability of cloud computing have enabled the solution of problem instances of significant size. In addition, the use of valid inequalities to strengthen mathematical formulations of MIRPs has increased the range of practical problems that can be tackled by means of commercial solvers (Agra,

* Corresponding author at: Departamento de Informática y Estadística, Universidad Rey Juan Carlos, C/ Tulipán s/n, 28933 Móstoles (Madrid), Spain.
E-mail addresses: sergio.cavero@urjc.es (S. Cavero), laguna@colorado.edu (M. Laguna), eduardo.pardo@urjc.es (E.G. Pardo).

https://doi.org/10.1016/j.cie.2024.109978
Received 24 July 2023; Received in revised form 3 January 2024; Accepted 8 February 2024
Available online 14 February 2024

Andersson, Christiansen, & Wolsey, 2013; Agra, Christiansen, & Delgado, 2013; Eide, Håhjem Årdal, Evsikova, Hvattum, & Urrutia, 2020; Engineer, Furman, Nemhauser, Savelsbergh, & Song, 2012; Grønhaug, Christiansen, Desaulniers, & Desrosiers, 2010; Sherali, Al-Yakoob, & Hassan, 1999). To incorporate characteristics that are specific to each setting, the literature includes custom solution methods. These procedures are based on heuristic approaches and decomposition techniques (Agra, Christiansen, Delgado, & Simonetti, 2014; Christiansen, 1999; Siswanto, Essam, & Sarker, 2011; Hiassat, Diabat, & Rahwan, 2017; Wang, Nuo, & Han, 2023).

Some of the features in the SSIRP that we address are captured in mathematical formulations that have appeared in the literature. For instance, Agra, Christiansen, and Delgado (2017) introduce two arc-based formulations where the time is discretized into periods. One formulation addresses the problem of variable consumption and the other one focuses on constant consumption rates in which inventory levels must be within specified bounds during the entire planning horizon. The time discretization in (Agra et al., 2017) deviates from the discrete time formulation in (Agra et al., 2013), where the mathematical formulation allows a ship to stay in a port for several time periods. It also deviates from the formulation in (Grønhaug & Christiansen, 2009), where each port operation is assumed to be completed within one period and sailing is modeled as an integer number of periods. Agra et al. (2017), in contrast, use periods that accommodate the maximum time for a full ship operation (e.g., loading and unloading) and may also include several port visits and sailing time. The length of the period must be chosen judiciously because restrictions such as port capacity are enforced during the entire period and inventory constraints are checked only at the end of the period.

The discrete time formulation in (Agra et al., 2013) models port operations in more detail than other formulations in the literature. In each period, a ship can either be waiting, performing a port operation (loading or unloading), or sailing. In this formulation, the time required for a port operation depends on the amount of product to load or unload. While there is no question that the level of detail goes beyond anything anyone else has proposed, the formulation is limited to a single product, which is unrealistic for most applications. It is worth mentioning that the literature also includes studies that address short sea inventory routing problems by considering a continuous-time planning horizon (Stanzani et al., 2018; Diz et al., 2017).

As mentioned above, a MIRP in a specific industry and company represents a unique challenge. In our case, we were confronted by the need to include heterogeneous vessels, multiple ports, a discrete-time planning horizon, and costs associated with port operations, vessels, and travel. These characteristics are relevant in the project that originated this research effort. In our review of the literature, we could not find models that consider the relevant features in combination with long-term planning horizons. This motivated the work presented here, contributing with another SSIRP version to the literature.

We introduce a path-based mathematical programming formulation specifically tailored to address the complexities of the SSIRP that we studied in the oil company. The formulation encompasses critical real-world aspects, such as managing varied and restricted supply chains, addressing multiple incompatible product types, and navigating the intricate dynamics of port operations. The path-based approach that we took reduces the complexity of the problem, offering a manageable methodology to obtain solutions to instances found in practice. We also created a matheuristic by combining heuristic strategies with a reduced mixed integer programming model. The matheuristic decomposes the problem into smaller subproblems, enabling separate optimization decisions associated with resource allocation and routing. This combination results in a procedure capable of handling problem instances that are larger than those that a commercial mixed-integer programming solver can tackle in a reasonable amount of computer time. The development of the matheuristic for the SSIRP that we faced is one of the main contributions of this work.

2. Problem description

The problem consists of creating a plan to transport clean oil products of a large company. The main decisions are vessel selection and assignment, and the amount of product to load/unload at each port in each period of a planning horizon. A subset of vessels must be selected to perform operations during a planning horizon. Each selected vessel is assigned to a sequence of segments (i.e., ordered sets of ports) to configure a path that the vessel will follow during the planning horizon. The amount of each product to load and/or unload at each port visited by the vessels must also be determined.

The primary objective of the problem is to formulate a cost-effective plan that minimizes the distribution costs. Particularly, the total cost of a schedule is calculated as the sum of several components:

- Fixed Cost: This is the cost associated with the utilization of a vessel. It is charged only if the resource is assigned to a path.
- Travel Cost: This cost depends on the path that a vessel follows. The cost considers the capacity and speed of the vessel, as well as the distance and duration of the path.
- Loading and Unloading Costs: These costs are associated with the amount of product to load and unload at supply and demand ports.
- Inventory Shortage Cost: This cost arises when there is insufficient inventory to meet the demand at a demand port. It is reflective of the spot price of the product, which is assumed to be significantly higher than the production cost.

The challenge of the problem lies in configuring a distribution plan that deals with fluctuating demand while maintaining specified inventory levels throughout a predetermined planning period. It is important to note that there is no need to treat this situation as a multi-objective problem given that the primary objective is total cost minimization.

The problem is modeled on a network of supply and demand ports. For all practical purposes, supply ports are considered to have unlimited capacity to produce a subset of the products. That is, supply ports are not able to produce all products, but they can supply unlimited amounts of the ones they produce. Vessels (also referred to as resources) with several compartments of various capacities transport products. Products consist of several types of fuels (e.g., blend grade, premium, ethanol, or jet fuel) that cannot be mixed. Resources travel from port to port, loading and unloading products, during a planning horizon.

Additional complexities associated with the real setting include draft restrictions at the ports and the transportation of dirty products with the corresponding cleaning decisions. While we addressed the draft restrictions by adjusting vessel capacities, the handling of dirty products was left for a subsequent phase of the project. We addressed the draft restrictions by adjusting vessel capacities according to the draft limitation of the ports that they visit. This ensures that the vessels can access all the ports in their paths without violating the draft constraints. In the case of dirty products, we note that this can be handled in a post-processing of a proposed solution, by adding cleaning operations and costs whenever a vessel switches from a dirty product to a clean product, or vice versa. Finally, we assumed that the compartments in each resource are such that any load is feasible, in the sense that all products can be separated in the quantities prescribed by the model. In other words, we modeled capacity at the resource level instead of the tank (i.e., compartment) level. This is an important consideration, given that products cannot be mixed, which means that without compartment flexibility a model with a higher level of specificity would be required.

A complete solution must stipulate the quantity of each product that will be delivered to each demand node in the path of a resource. The fleet is heterogeneous (in terms of capacity and speed) and therefore the cost of delivering products depends on both the route and the resource. In addition to resource capacity, a solution must meet minimum and maximum inventory levels for each product, at each demand port, as

well as dock capacity at all ports. As mentioned above, in the context that we examined, the relevant costs include production, purchasing, port visit, sailing, loading/unloading, and inventory shortages.

3. Mathematical programming formulation

Our short-sea inventory routing problem formulation uses the concept of path segments. A segment is an ordered sequence of ports that represents a subset of the ports that a resource will visit during the planning horizon. A path is an ordered set of segments. The first port in the path is the resource's current port. A path may terminate before the planning horizon is over, indicating that the resource is not sailing during the entire horizon. Nodes in the network may be demand ports, supply ports, or bifunctional (i.e., both demand and supply). Fig. 1 shows and example with two segments in a planning horizon of 10 periods.

The first segment (blue arcs) consists of a sequence from the origin to node 1 and then node 2. This is followed by a sequence shown with red arcs that visits nodes 3, 4, 5, where the path terminates. Our approach to the short-sea inventory routing problem consists of concatenating segments to form a path for each resource that has been chosen to sail. The use of segments gives an analyst full flexibility. For example, the set of segments may include all pairs of ports (i.e., a fully connected network). However, in practice it is not necessary to include all port pairs because vessels tend to travel in a specific direction (e.g., following ports along the Florida peninsula). A carefully curated list of segments to include in the model is an important decision left to the analyst.

In addition, the formulation includes loading and unloading decisions. The main constraints enforce resource and port capacity as well as inventory flows, both within each resource and at the ports. The objective is to minimize total cost, which includes fixed costs for using resources during the planning horizon, travel costs, loading and unloading costs, and inventory shortage costs.

The following mixed-integer programming formulation of the problem assumes that a preprocessing step has been performed to create a list of path segments with their associated cost.

Sets

I: set of products (indexed by i).
J: set of ports (indexed by j).
J_i^l: set of supply ports for product $i \in I$, i.e., all ports where product i can be loaded.
J_i^u: set of demand ports for product $i \in I$ i.e., all ports where product i can be unloaded. Port j is bifunctional if $j \in J_i^l \cap J_i^u$.
J_p: set of ports in segment $p \in P$.
P: set of segments, where segment $p \in P$ is an ordered list of ports with p_{first} representing the index of the port in the first position of the segment and p_{last} is the index of the port at the end of the segment.
PQ: set of compatible segment pairs (p, q), where p and q are compatible if $p_{last} = q_{first}$.
R: set of resources (index by r).
T: set of periods in the planning horizon, where the first period is 1 and the last is $|T|$.

Parameters

b_r: total capacity of resource $r \in R$. It is estimated as the sum of the capacities of all compartments of a vessel.
c_{rp}^a: cost of assigning resource $r \in R$ to segment $p \in P$. This cost includes travel and port costs.
c_r^f: fixed cost of using resource $r \in R$ in the planning horizon.

(continued on next column)

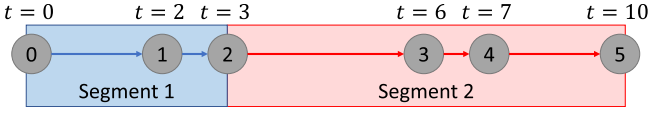

Fig. 1. Path with two-segments in a 10-period planning horizon.

(continued)

c_{ijrt}^l: cost of loading a barrel of product $i \in I$ at port $j \in J_i^l$ on resource $r \in R$ in period $t \in T$. This parameter includes the cost of producing a barrel of product $i \in I$, which depends on the period $t \in T$ when the product is loaded.
c_{ijr}^u: cost of unloading a barrel of product $i \in I$ at port $j \in J_i^u$ from resource $r \in R$. This parameter, unlike the cost of loading, does not depend on the period.
c_{ijt}^-: per barrel cost of inventory shortage of product $i \in I$ at port $j \in J_i^u$ in period $t \in T$. This parameter is indexed by period because these values reflect the spot price (per barrel) to satisfy inventory shortages. We assume that the spot price is significantly larger than the production cost.
d_{ijt}: demand for product $i \in I$ at port $j \in J_i^u$ in period.$t \in T$
D_j: number of available docks in port.$j \in J$
p^r: segment representing the node where resource $r \in R$ is at the beginning of the planning horizon. This may be interpreted as the "source" path for r.
s_{ij}^{min}: minimum stock level of product $i \in I$ at port.$j \in J_i^u$
s_{ij}^{max}: maximum stock level of product $i \in I$ at port.$j \in J_i^u$
S_{ijt}: supply limit of product $i \in I$ at port $j \in J_i^l$ in period.$t \in T$
τ_{jrp}: time required by resource $r \in R$ to reach port $j \in J$ when following segment.$p \in P$

Continuous variables

l_{ijrt}: amount of product $i \in I$ loaded to resource $r \in R$ at port $j \in J_i^l$ in period $t \in T$.
u_{ijrt}: amount of product $i \in I$ unloaded from resource $r \in R$ at port $j \in J_i^u$ in period $t \in T$.
x_{irt}: amount of product $i \in I$ in resource $r \in R$ in period $t \in T$.
s_{ijt}: amount of product $i \in I$ in stock at port $j \in J_i^u$ in period $t \in T$.
s_{ijt}^-: shortage of product $i \in I$ at port $j \in J_i^u$ in period $t \in T$.

Binary variables

y_{rpqt}: equals 1 if resource $r \in R$ transitions from segment $p \in P$ to segment $q \in P$ in period $t \in T$. These variables are defined for $(p, q) \in PQ$ only.

Formulation

Minimize

$$\sum_{r\in R}\sum_{q:(p^r,q)\in PQ} c_r^f y_{rp^rq1} + \quad (1)$$
$$\sum_{r\in R}\sum_{(p,q)\in PQ}\sum_{t\in T} c_{rq}^a y_{rpqt} + \quad (2)$$
$$\sum_{i\in I}\sum_{j\in J_i^l}\sum_{r\in R}\sum_{t\in T} c_{ijr}^l l_{ijrt} + \quad (3)$$
$$\sum_{i\in I}\sum_{j\in J_i^u}\sum_{r\in R}\sum_{t\in T} c_{ijr}^u u_{ijrt} + \quad (4)$$
$$\sum_{i\in I}\sum_{j\in J_i^u}\sum_{t\in T} c_{ijt}^- s_{ijt}^- \quad (5)$$

Subject to

Port activity constraints

$$x_{irt} = x_{ir,t-1} + \sum_{j\in J_i^l} l_{ijrt} - \sum_{j\in J_i^u} u_{ijrt} \quad \forall i \in I, r \in R, t \in T \quad (6)$$
$$\sum_{j\in J_i^u} u_{ijrt} - x_{ir,t-1} \le 0 \quad \forall i \in I, r \in R, t \in T \quad (7)$$
$$\sum_{i:j\in J_i^l} l_{ijrt} \le b_r \sum_{(p,q)\in PQ}\sum_{t':t=t'+\tau_{jrp}} y_{rpqt'} \quad \forall j \in J_i^l, r \in R, t \in T \quad (8)$$
$$\sum_{i:j\in J_i^u} u_{ijrt} \le b_r \sum_{(p,q)\in PQ}\sum_{t':t=t'+\tau_{jrp}} y_{rpqt'} \quad \forall j \in J_i^u, r \in R, t \in T \quad (9)$$

Path flow and dock capacity constraints

$$\sum_{q:(p^r,q)\in PQ} y_{rp^rq1} \le 1 \quad \forall r \in R \quad (10)$$
$$\sum_{q:(q,p)\in PQ} y_{rqp,t-\tau_{p_{last}rp}} \ge \sum_{q:(p,q)\in PQ} y_{rpqt} \quad \forall p \in P, r \in R, t \in T \quad (11)$$
$$\sum_{r\in R}\sum_{(p,q)\in PQ\wedge j\in J_q}\sum_{t':t=t'+\tau_{jrp}} y_{rpqt'} \le D_j \quad \forall j \in J, t \in T \quad (12)$$

Inventory and resource capacity constraints

$$s_{ijt} = s_{ijt-1} + s_{ijt}^- + \sum_{r\in R} u_{ijrt} - \sum_{r\in R} l_{ijrt} - d_{ijt} \quad \forall i \in I, j \in J_i^u, t \in T \quad (13)$$
$$s_{ij}^{min} \le s_{ijt} \le s_{ij}^{max} \quad \forall i \in I, j \in J_i^u, t \in T \quad (14)$$
$$\sum_{i\in I} x_{irt} \le b_r \quad \forall r \in R, t \in T \quad (15)$$

Nonnegativity and binary restrictions

$$l_{ijrt} \ge 0 \quad \forall i \in I, j \in J_i^l, t \in T \quad (16)$$
$$u_{ijrt} \ge 0 \quad \forall i \in I, j \in J_i^u, t \in T \quad (17)$$
$$x_{irt} \ge 0 \quad \forall i \in I, t \in T \quad (18)$$
$$s_{ijt} \ge 0 \quad \forall i \in I, j \in J_i^u, t \in T \quad (19)$$
$$s_{ijt}^- \ge 0 \quad \forall i \in I, j \in J_i^u, t \in T \quad (20)$$
$$y_{rpqt} = \{0, 1\} \quad \forall r \in R, (p, q) \in PQ, t \in T \quad (21)$$

Interpretation of the model

(1) Total resource fixed cost. If resource r is assigned to segment q in period 1 then $y_{rp^rq1} = 1$ and the fixed cost c_r^f is charged.

(2) Total segment assignment cost. If resource r transitions from segment p to q in period t ($y_{rpqt} = 1$), then the cost c^a_{rq} of resource r traversing segment q is charged.
(3) Total loading cost. The loading cost c^l_{ijr} is charged to the l_{ijrt} barrels of product i loaded at port j onto resource r in period t.
(4) Total unloading cost. The unloading cost c^u_{ijr} is charged to the u_{ijrt} barrels of product i unloaded at port j from resource r in period t.
(5) Total inventory shortage cost. Inventory shortages occur when the demand exceeds the supply in a period. The shortage amount is given by s^-_{ijt} and the unit shortage cots is c^-_{ijt}.
(6) The amount of product i on resource r after visiting port j in period t is equal to the amount in the previous period plus the amount loaded minus the amount unloaded at port j. This constraint set may be interpreted as an inventory flow equation for each product on each resource.
(7) The amount of product i that can be unloaded at port j from resource r in period t cannot exceed the amount of that product that is on the resource in the previous period.
(8) Loading of any product onto resource r at port j in period t can only occur if resource r is at port j in period t. That means that resource r must be assigned to segment p in period $t - \tau_{jrp}$, where τ_{jrp} is the number of periods that resource r needs to reach port j when following segment p.
(9) Same as (8) for unloading activities.
(10) Each resource r may make the transition from its current port p^r to a compatible segment q in the first period of the planning horizon. If that transition is not made, then the resource is not sailing, and the fixed cost is not charged (see (1) in the objective function).
(11) This a conservation of flow constraint, where a resource r that finishes segment p in period t within the planning horizon must transition to a segment q.
(12) This set of constraints enforce docking capacity at port j. All resources that made the transition from segment p to a segment q (that includes port j) in period $t - \tau_{jrq}$ will arrive at port j in period t. There cannot be more than D_j resources arriving at port j in any period.
(13) Inventory flow constraints for all products at all ports. The inventory in period t is equal to the inventory in the previous period plus any unloaded amounts minus any amounts loaded minus the demand plus any shortages.
(14) Inventory of any product at any port should be within desired limits during the entire planning horizon.
(15) The total amount of product on a resource cannot exceed the resource capacity.

Segments and paths play a key role in our model. The solution process includes a pre-processing step that generates PQ, the set of compatible segments (i.e., (p, q) pair where the final port in p is the first port in q). The model uses the binary variables y to represent whether a resource makes the transition from one segment to another in each period. If a y variable is set to one, it indicates that the resource will commence its journey from the current segment to the next within the specified period. Once a resource completes a segment, the model must decide whether the resource should transition to another segment or not, and which segment will be the next one. It is important to note that a resource cannot remain idle at a port unless it is the last port of a segment. Therefore, the path of resource is given by the y variables that are set to one in the solution of the model.

In brief, our model tackles the SSIRP by selecting vessels from a list and assigning them to compatible port segments with the objective of minimizing the total cost. It takes into consideration the fixed costs of each vessel as well as the costs of travelling, loading, unloading, and inventory shortage. The shared resources of vessels are constrained by their capacity and availability, and the model ensures that the dock capacity at each port is not exceeded. The solution of the proposed model must deal with several challenges, namely, 1) handling many variables and constraints when considering long planning horizons and a large network of ports and multiple vessels, 2) combining discrete and continuous decision variables, and 3) meeting variable demand and inventory limits. All these considerations play a significant role in the ability of a commercial solver to find feasible solutions and confirm optimality of the solutions that it finds.

4. Matheuristic approach

The idea behind our matheuristic is to decompose the model into two sets of decisions. The first set deals with the selection of resources and their assignment to segments to build complete paths. The second set determines, for each resource, the amounts of each product to load and unload at the ports that the resource visits. The order and timing of the port visits are determined by the segment assignments. A solution is represented as a list of compatible segments for each resource:

$$\Pi = \{\pi(1), \ldots, \pi(|R|)\}$$

Where $\pi(r)$ is the path for resource r, which consists of an ordered list of n_r compatible segments that starts at p^r:

$$\pi(r) = (p^r, q_r(1), \ldots, q_r(n_r))$$

Where $q_r(k)$ is the k^{th} segment in the path of resource r such that $(p^r, q_r(1)) \in PQ$ and $(q_r(k-1), q_r(k)) \in PQ$ for $k = 2, \ldots, n_r$. A value of $n_r = 0$ indicates that resource r is not being used during the current planning horizon. Therefore, the fixed costs associate with a solution Π are:

$$\text{fixed\_cost}(\Pi) = \sum_{r:n_r>0} c^f_r$$

The travel costs depend on the segment assignments and can be calculated as follows:

$$\text{travel\_cost}(\Pi) = \sum_{k=1}^{k=n_r} c^a_{rq_r(k)}$$

For a solution Π, it is possible to determine where each resource is going to be in each period of the planning horizon. Let $H^j_t(\Pi)$ be the set of resources at port j in period t for the set of paths Π. Solution Π must be feasible with respect to (12), the dock capacity at all ports. Port capacities are observed if the number of resources at a port in each period does not exceed the dock capacity of the port, that is:

$$\left|H^j_t(\Pi)\right| \le D_j \forall j \in J, t \in T$$

The operational costs of loading and unloading product as well as inventory shortages associated with a solution Π are calculated by the solution of the following linear program:

Minimize operational_cost(Π) = (3) + (4) + (5).
Subject to (6), (7), and (13)–(20)

$$\sum_{i:j\in J^l_i} l_{ijrt} \le \begin{cases} b_r & \text{if } r \in H^j_t(\Pi) \\ 0 & \text{otherwise} \end{cases} \quad \forall j \in J^l_i, r \in R, t \in T \tag{22}$$

$$\sum_{i:j\in J^u_i} u_{ijrt} \le \begin{cases} b_r & \text{if } r \in H^j_t(\Pi) \\ 0 & \text{otherwise} \end{cases} \quad \forall j \in J^u_i, r \in R, t \in T \tag{23}$$

The linear program decides how much to load and unload of each product to minimize the operational cost associated with the set of paths Π. Note that only the right-hand-side (RHS) values of constraints (22) and (23) must be changed to find the operational cost of Π. Therefore, after solving the linear program the first time, the evaluation of additional solutions can be achieved by changing the RHS values of (22) and (23) and resolving starting from the current solution. This is a fast

process for commercial mathematical programming solvers such as Gurobi. The total cost of solution Π is given by:

$$\text{total\_cost}(\Pi) = \text{fixed\_cost}(\Pi) + \text{travel\_cost}(\Pi) + \text{operational\_cost}(\Pi) \quad (24)$$

Our matheuristic searches the space of segment assignments to find the set of paths Π^* with the minimum total cost.

4.1. Construction

A solution Π may be constructed by adding one segment at a time starting from an empty solution for which $n_r = 0$ for all $r \in R$. An empty solution (that is, a solution with no paths) has zero fixed and travel costs. The operational cost for such a solution is given by the inventory shortages. The most attractive segments to add are those that provide the best tradeoff between reducing inventory shortage costs versus adding travel and loading/unloading costs. Considering the cost of the solution after assigning a resource to a segment, a greedy criterion can be defined for the selection of segments. Specifically, at each iteration, the procedure determines which segment minimizes the solution cost and adds it to the partial solution. The evaluation of the solution cost implies the computation of the simplified MIP to calculate the total cost (24). This construction can be fully deterministic or based on a semi-greedy approach as typically done in GRASP (Feo & Resende, 1995). The construction process stops when adding more segments does not reduce the total cost of the current solution. At that point, those resources that have been assigned to any path are considered as selected.

Algorithm 1 is the general scheme of the proposed construction process. The constructive procedure operates on the input data (D) and uses the value of the α parameter to control the randomness/greediness of the construction. The procedure starts with an empty solution (line 1) and a candidate list of segments (line 2). The resources are sorted (line 3) and a path is generated for each resource (lines 4–20). While the solution improves (line 7), the procedure evaluates the potential addition of a segment to the solution. The operational cost of the solution is calculated by solving the linear programming model (line 9) while *total_cost*(Π) corresponds to equation (24). The maximum and minimum values of the current objective function value are recorded (lines 8 and 9). These values are used to establish a threshold for generating a Restricted Candidate List (RCL). A candidate from the RCL is randomly selected and added to the path. If the addition of a segment deteriorates the current solution (lines 18 and 19), the construction ends (line 21). A resource with an empty path indicates that the resource has not been selected.

Algorithm 1. General scheme of GRASP constructive procedure.

```
Procedure Construction (D, α)
1. Π←∅
2. CL←PQ
3. R'←sort(R)
4. For r ∈ R:
   5. improvement←True
   6. While improvement :
      7. improvement←False
      8. g_min ← min_{s∈CL} g(Π, r, s)
      9. g_max ← max_{s∈CL} g(Π, r, s)
      10. μ←g_min + α(g_max − g_min)
      11. RCL←{s ∈ CL : g(Π, r, s) ≤ total_cost(Π) ≤ μ}
          /* total_cost(Π) calls the MIP model to evaluate the solution */
      12. If RCL ≠ ∅:
          13. s←random(RCL)
          14. add(Π, r, s)
          15. improvement←True
      16. End If
   17. End While
   18. If π(r) is empty
          19. Break
   20. End For
   21. Return Π
```

An important aspect to consider in the design of the constructive procedure is the time required to build a solution. Determining the benefit of adding a segment entails a full evaluation of the partial solution. If we want to determine the best segment to add to the current set of paths, the calculation must consider all segments and all possible resources. Preliminary experimentation revealed that the time associated with such an exact calculation of benefit at each step of the construction process is prohibitively long. This is why in Algorithm 1 we ranked the resources according to a criterion that considers the relationship between vessel cost and capacity. The procedure selects resources one at a time to determine complete paths. As shown above, the construction ends when no segment is assigned to the selected resource.

4.2. Neighborhood search

We developed five neighborhood searches to be used in the solution improvement phase of GRASP. At each iteration of a neighborhood search, the current solution is replaced by a neighbor solution. The set of neighbor solutions is referred to as the "neighborhood", and the replacement of the solution is denoted as a "move". Out of the five neighborhoods, four utilize segments as the basic elements for the move operator, with one focusing on operations involving resources.

The segment-based neighborhoods in our procedure are insert, swap, replace, and remove. The fundamental moves behind these neighborhoods are *add* and *drop*.

An *add* move adds a segment to the existing path of a resource. Formally, we denoted the operator as $add(r,p,\rho)$, where p is the segment to be added in position ρ of an existing path $\pi(r) = (p^r, q_r(1), \ldots, q_r(n_r))$. Then, the path $\pi'(r) = (p^r, q_r'(1), \ldots, q_r'(n_r'))$ after the addition is such that:

$$q_r'(k) = q_r(k) \text{ for } k = 1, \ldots, \rho - 1$$

$$q_r'(\rho) = p$$

$$q_r'(k+1) = q_r(k) \text{ for } k = \rho, \ldots, n_r$$

$$n_r' = n_r + 1$$

A feasible add move is such that $(q_r(\rho-1), p) \in PQ$ and $(p, q_r(\rho)) \in PQ$.

A *drop* move eliminates a segment from the existing path of a resource. Let, $drop(r,\rho)$ be the operator that deletes the segment in position ρ of an existing path $\pi(r) = (p^r, q_r(1), \ldots, q_r(n_r))$. Then the path $\pi'(r) = (p^r, q_r'(1), \ldots, q_r'(n_r'))$ after the deletion is such that:

$$q_r'(k) = q_r(k) \text{ for } k = 1, \ldots, \rho - 1$$

$$q_r'(k) = q_r(k+1) \text{ for } k = \rho, \ldots, n_r - 1$$

$$n_r' = n_r - 1$$

A feasible *drop* is such that $(q_r(\rho-1), q_r(\rho+1)) \in PQ$. Deleting a path is particularly attractive for reducing operational and travel costs. Of particular benefit is the removal of segments with no load or unload operations in the ports that it visits, or when these operations could be done by other resources.

We use add/drop moves to create the *insert*, *swap*, *replace*, and *remove* neighborhoods. The insert neighborhood consists of all solutions obtained by removing a segment from its current position and adding it to a different position in the path. Similarly, the swap neighborhood consists of exchanging the positions of two segments in a path. This is achieved by removing the segments and then adding them to their swapped positions. The replace neighborhood consists of all solutions obtained by exchanging a segment in a current path with a segment that is not currently in the solution. The difference between the swap and replace neighborhoods is that the swap neighborhood modifies the positions of

two segments of the path while the replace neighborhood modifies just one. Finally, the remove neighborhood consists of all solutions obtained by dropping one segment from a path.

Fig. 2 illustrates the four moves that we use in our search. Fig. 2a shows an insert move where segment 4 (denoted as p_4), which starts at $t = 8$, is inserted at $t = 3$. Fig. 2b illustrates a swap move where segments 2 (denoted as p_2) and 4 are swapped. Fig. 2c provides an example of a replace move where segment 2 is replaced by segment 4. Lastly, Fig. 2d shows a remove move where segment 3 (denoted as p_3) is removed from the route.

So far, we have described four neighborhoods that are based on segment operations. We now introduce a move that focuses on resources with the goal of reducing the fixed cost of the solution. We refer to this operation, which involves exchanging the paths of two resources, as a *substitute* move. Let $\pi(r) = (p^r, q_r(1), \ldots, q_r(n_r))$ be the path of resource r, and let $\pi(r') = (p^{r'}, q_{r'}(1), \ldots, q_{r'}(n_{r'}))$ be the path of resource r'. Then, the paths for r and r' after the substitute move are such that $\pi(r) = (p^{r'}, q_{r'}(1), \ldots, q_{r'}(n_{r'}))$ and $\pi(r') = (p^r, q_r(1), \ldots, q_r(n_r))$. In this case, a feasible substitution is such resources r and r' share the starting point of the route, i.e., $p^r = p^{r'}$. It is important to note that the substitute move allows $\pi(r) \neq \emptyset$ or $\pi(r') = \emptyset$. This means that a resource currently in the solution can be replaced by another resource that is not currently in the solution.

The proposed neighborhoods could result in a path that extends beyond the end of the planning horizon, i.e., beyond period $|T|$. However, generally, those paths will not be attractive due to their cost and the lack of benefit for covering demand that may occur after the planning horizon ends. The local search could start from a solution with shortages. However, because shortages are heavily penalized, moves from solutions with shortages will favor search directions where shortages decrease. Once a solution without shortages is reached, the search does not consider moves that would result in shortages.

We consider two neighborhood exploration strategies:

11. Complete neighborhood exploration. This means that all possible moves are explored to find the best in terms of improving (in the local sense) the objective function value.
21. Frist-improving sampling. This strategy systematically explores a neighborhood and terminates as soon as an improving move is found or when the entire neighborhood is explored without finding an improving direction.

We tested both strategies within a local search scheme in our preliminary experimentation.

4.3. General procedure

We combined the solution construction process in Section 4.1 and the neighborhood searches in Section 4.2 to configure a Greedy Randomized Adaptive Search Procedure (GRASP). We assembled the neighborhood searches in the framework of a Variable Neighborhood Descent (VND) (Duarte, Sánchez-Oro, Mladenović, & Todosijević, 2018). VND is a search procedure that performs a systematic exploration of different neighborhoods around the current solution, seeking to improve the objective function value of the current solution. VND moves from the current solution to a neighbor solution if the objective function value is improved and the search process continues from the new solution. If no such a solution is found, the search moves to a different neighborhood and repeats the process until no further improvement can be found. Our basic VND sequentially explores all neighborhoods in an order that was determined using an automated fine-tuning platform, as described in the following section. Both GRASP and VND, whether used in combination or in isolation, have been reported to perform well in combinatorial optimization problems that are similar to ours (Ronconi & Manguino, 2022; Sanghikian, Martinelli, & Abu-Marrul, 2021; Diz, Hamacher, & Oliveira, 2021). Their efficiency and robustness make them suitable frameworks for our problem setting.

Algorithm 2 shows the pseudocode of our GRASP-VND implementation. The iterative procedure operates on the input data for a maximum amount of computer time (*max_time*) and uses α to control the semi-greedy nature of the GRASP construction.

Algorithm 2. General scheme of GRASP constructive procedure.

```
Procedure GRASP-VND(D, max_time, α)
1. Π* ← ∅
2. While elapsed time < max_time:
3.   Π ← Construction(D, α)
4.   Π' ← VND(Π)
5.   If total_cost(Π') ≤ total_cost(Π*)
       /* total_cost calls the MIP model to evaluate the solution */
6.     Π* ← Π'
7. End While
8. Return Π
```

The procedure starts by initializing the best solution as empty (line 1). At each iteration, a new solution Π is generated using the Construction procedure with the input data D and the value of α (line 3). This solution is then subjected to the improvement process of the VND procedure, resulting in a potentially improved solution Π′ (line 4). If the

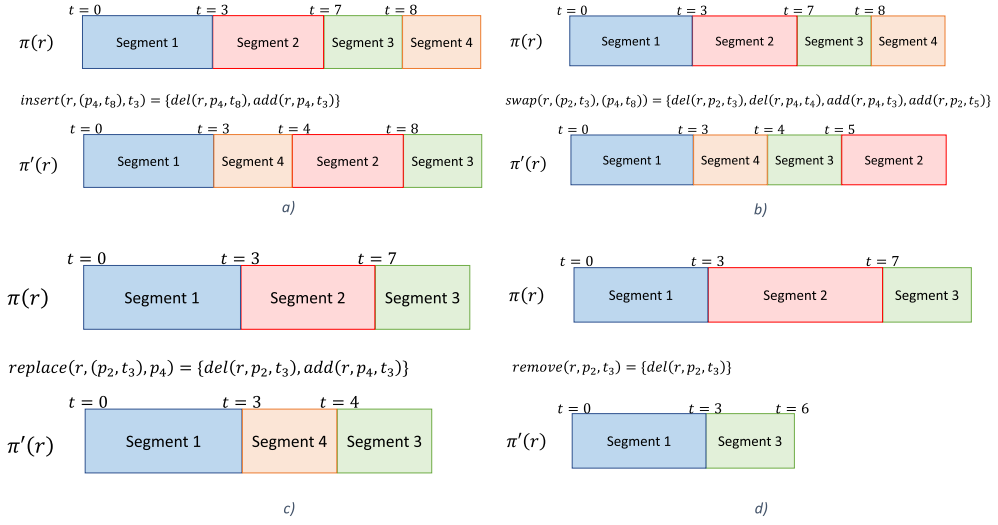

Fig. 2. Example of the moves based on segments: a) insert move, b) swap move, c) replace move, and d) remove move.

total cost of Π', calculated by solving the simplified linear program (24), is lower than the total cost of the current best solution Π^* (line 5), then Π' becomes the new best solution (line 6). The process repeats until the maximum time allowed for execution is reached (line 7). The algorithm returns the best solution found (line 8).

5. Computational experiments

This section presents the results of our computational experiments designed to evaluate the effectiveness of our proposed solution procedure. Section 5.1 details the preliminary experiments that calibrate the parameters of our matheuristic and in Section 5.2 we compare and discuss the results obtained by both the mathematical model and the matheuristic.

Before delving into the experimentation, it is important to describe the data set used in our research. We utilized real-world instances provided by the company for which we did this work and generated artificial instances by modifying certain parameters according to specific research needs. The data set considers 4 products, 12 resources, 6, 9 and 12 ports and a planning horizon of either 25, 50 or 75 days. Demand is varied by setting a demand factor δ to either 1, which represents the base case provided by the company, or 1.15, which represents an increase of 15% in demand at each port. These parameters were combined to create 10 instances for each combination of number of ports, length of planning horizon, and demand factor. This results in 180 instances. We selected 6 representative instances to configure a training set for the preliminary experimentation.

All experiments were conducted on an AMD EPYC 7282 8-core virtual CPU with 16 GB of RAM, running Ubuntu 20.04.2 64-bit LTS. Our algorithms were implemented using Python 3 and Gurobi 10. It is important to note that while Gurobi is designed to utilize all available cores, Python is single-threaded and therefore does not fully utilize the computational capacity of our system when running the matheuristic approach.

5.1. Preliminary experiments

To test the various configurations of the components of the matheuristic procedure and to illustrate the proposed strategies, we conducted several preliminary experiments over the training set. We first tested different values of the parameter (α) used to control the greediness and randomness of the constructive procedure. Then, we examined the improvement in solution quality provided by each local search separately. Finally, we compared those results with the improvement provided by combination of all local searches within the VND framework.

The results of the first experiment, which tests the α values, are presented in Table 1. The table shows the average cost of the solutions, the CPU time, and number of best solutions obtained for different values of α (i.e., 0.1, 0.2, 0.3, 0.4, 0.5 and a random value in the range [0.1, 0.3]) and for constructing 1, 5, and 10 solutions. When using a range instead of a fixed value, the α value is sampled from the given range at each step of the construction process (see step 10 in Algorithm). The best solution for each instance is the one found over all the runs associated with this experiment.

The values in Table 1 reveal that a random value of α between 0.1 and 0.3 results in the lowest cost when constructing 5 and 10 solutions. This suggests that introducing some randomness in the selection of α seems to lead to better solutions. As expected, we can observe the sampling effect of GRASP constructions. That is, the cost generally decreases as the number of constructions increases. We also observe that the increase in CPU time is linear with respect to the number of constructions and that it does not seem to depend on the value of α. The increase in computational effort is worth noting to limit the number of constructions when dealing with large instances.

Table 1
Performance of alpha values in the constructive procedure for 1, 5, and 10 solutions.

α	Metric	1 solution	5 solutions	10 solutions
[0.1, 0.3]	Total cost	9.02E+07	1.47E+07	1.35E+07
	CPU time (s)	43.10	215.73	424.30
	# Best	0	2	4
0.1	Total cost	1.31E+08	1.47E+07	1.43E+07
	CPU time (s)	42.68	217.55	431.50
	# Best	2	1	0
0.2	Total cost	1.32E+08	1.51E+07	1.47E+07
	CPU time (s)	44.87	218.61	439.12
	# Best	0	1	1
0.3	Total cost	1.33E+08	2.56E+07	1.43E+07
	CPU time (s)	45.00	221.05	432.05
	# Best	1	1	0
0.4	Total cost	1.56E+07	1.48E+07	1.44E+07
	CPU time (s)	44.40	217.36	434.90
	# Best	2	1	1
0.5	Total cost	2.74E+07	2.29E+07	1.41E+07
	CPU time (s)	45.01	219.67	441.65
	# Best	0	0	0

In our second experiment, we aimed to evaluate the effectiveness of the local search procedures and the VND framework both in terms of cost and computational time. The results of this comparison are presented in Table 2. For each local search, the table shows the cost components of the objective function as a percentage of the cost in the initial solution. For instance, the total cost of the VND solution is 27.4% of the total cost of the initial solution. The table also shows the computational effort expressed in CPU seconds.

As expected, the local searches that focus on path segments can reduce the assignment and shortage costs but have a difficult time reducing the fixed costs. The local search based on substitute moves reduces the fixed cost by either replacing or reducing resources. The effectiveness of combining the various neighborhood searches is shown by the results under the VND framework. Note that VND achieves the largest cost reduction in all categories and is the only procedure that eliminates shortages. The computational time of VND is significantly higher than the basic local searches. However, in our judgment, the quality of the VND solutions justifies the increased computational effort.

As part of our preliminary experiments, we used the *irace* software (López-Ibáñez, 2016) and the training set of instances to find the most effective configuration of our procedure. Iterated Racing for Automatic Algorithm Configuration (irace) is a software tool for automatically tuning the parameters of optimization algorithms. It is based on the idea of iterated racing, where a set of configurations (parameter settings) for an algorithm are repeatedly evaluated on a small subset of instances, and the best-performing configurations are selected for further

Table 2
Comparison of the solution quality and CPU time for the proposed local search procedures and VND methodology.

Local search	Total cost (%)	Fixed cost (%)	Assignment cost (%)	Shortage cost (%)	CPU time (s)
Insert	66.69	100.00	98.74	50.34	5.12
Swap	34.08	100.00	98.53	1.65	24.98
Replace	65.32	100.00	90.89	48.96	102.76
Remove	96.30	98.89	88.43	95.89	10.08
Substitute	53.05	88.25	97.20	34.92	45.12
VND	27.40	83.79	78.46	0.00	303.78

evaluation on a larger set of instances. Specifically, we used *irace* to determine the best order to consider the neighborhood searches within VND and to choose the best neighborhood exploration strategy (i.e., first or best improvement). The final configuration of our procedure and the one we used for the experiments reported in the next subsection is:

- Randomly select α between 0.1 and 0.3 at each step of the construction procedure
- Use VND as the local search by exploring the neighborhoods in the order substitute, replace, swap, insert, and remove.
- Use N2 as the neighborhood exploration strategy for all neighborhoods except replace, for which N1 is used.

As part of or preliminary experimentation, we wanted to understand the impact of the key model parameters on the effectiveness of the MIP formulation. To study the model's efficacy, we vary one parameter at a time from the base case of $|R| = 12$ resources, $|J| = 6$ ports, $|T| = 25$ days, and a demand factor $\delta = 1$. This allowed us to identify which parameters have the greatest impact on the model's performance.

Fig. 3 summarizes the results of this experiment in four charts. The charts depict the impact of changes in the number of ports (Fig. 3a), the periods in the planning horizon (Fig. 3b), the number of resources (Fig. 3c), and the demand factor (Fig. 3d). The solid blue lines show the changes in the CPU time (in seconds). The time values are shown on the left vertical axis, with an upper limit of 1 h (i.e., 3600 s). The orange dashed lines show the changes in the optimality gap, as reported by Gurobi. The optimality gap values are shown on the right vertical axis. The base case (i.e., of $|R| = 12, |J| = 6, |T| = 25, \delta = 1$) is solved to optimality (i.e., with an optimality gap of zero) in 731.5 s of computer time. An increase in the number of ports from 6 to 9 causes an increase in the optimality gap from 0 to 0.3 by the time the optimization process reaches the 1-hour limit. When the number of ports is increased to 12, the gap balloons to 0.45. We also observe large increases in the optimality gap as the number of days in the planning horizon increases from the base case of 25 to 50 and then 75. The gaps are 0.38 and 0.56, respectively. Fig. 3c shows that the model is practically insensitive to changes in the number of resources. All cases are solved to optimality within the allotted computational time. When varying the demand factor, we observe that for demand factors between 0.5 and 1, the MIP model can optimally solve the problem within the time limit. However, for the higher demand values associated with $\delta = 1.15$, the CPU time reaches the maximum limit, and the optimality gap increases to 0.19. We were not able to test larger δ values because the MIP model was unable to find an integer solution within the specified time limit.

For optimization runs with large optimality gaps we collected data to analyze the behavior of the solution process. We were interested in observing the trend of the values of the upper and lower bounds as reported by the optimizer.

Fig. 4 shows the cost of the incumbent solution (Upper bound), the cost of the best bound (Lower bound), and the optimality gap (GAP) reported during a Gurobi run of the MIP model for a representative instance. The horizontal axis in Fig. 4 represents the time in seconds, the left vertical axis represents the value of the bounds, and the right vertical axis represents the value of the optimality gap. To illustrate our point, we have zoomed in on the left vertical axis and therefore the origin is not zero. The dark green line represents the evolution of the cost of the

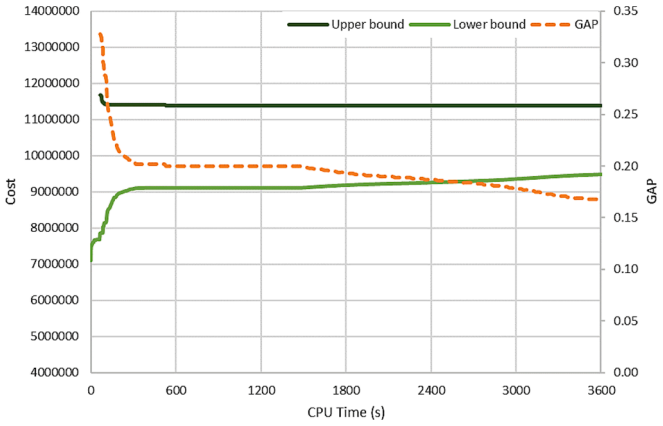

Fig. 4. Evolution of the best solution found, the best bound and the GAP during the execution time.

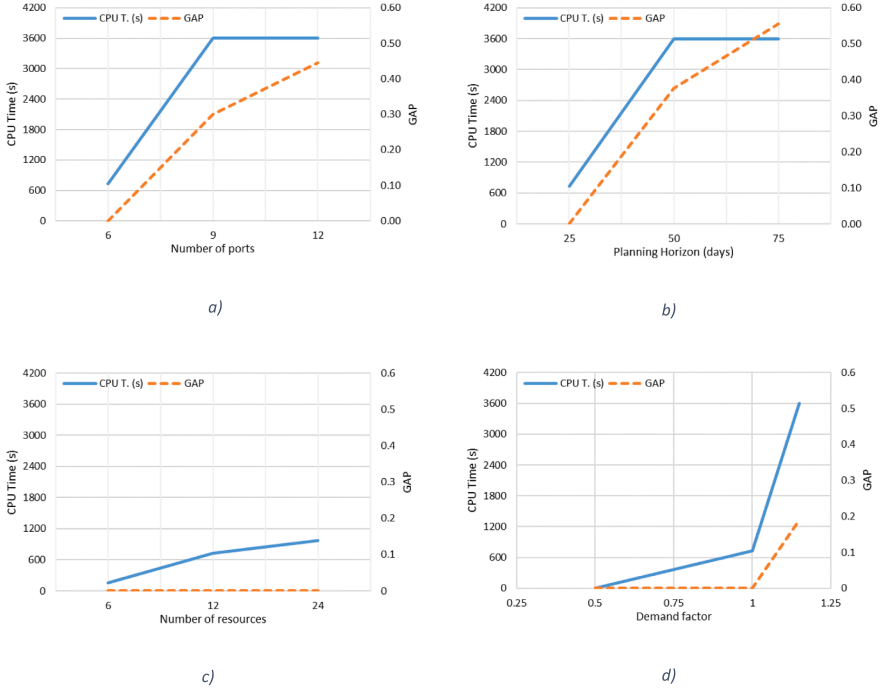

Fig. 3. Impact of varying one of the parameters on the performance of the MIP model.

incumbent solution (i.e., the upper bound), which starts at a large value (not shown on the chart) and quickly settles on a value that experiences a small relative improvement over the remaining time of the optimization run. Specifically, the optimizer finds a solution with a cost of 11,424,000 within the first two minutes of the run and, after one hour of the branch-and-bound search, the solution improves to 11,397,000. This represents an improvement of less than 1%. The light green line represents the value of the lower bound, which over the same period of one hour experiences a larger relative change than the upper bound. This means that most of the reduction of the optimality gap is due to the increase of the lower bound as opposed to any significant improvement of the incumbent solution. A conjecture from the extrapolation of this trend seems to indicate that the quality of the incumbent solution may be better than the 83.3% of optimality measured by the lower bound found at the end of the one-hour run. We have observed this behavior in all other representative instances in our data set.

Overall, this experimental analysis revealed several insights about our MIP model's performance. The variations of the values of the various parameters, such as number of ports, planning horizon, resources, and demand, notably impact the performance of the model. While the model showed higher sensitivity to certain parameters such as the number of ports or the planning horizon, it exhibited relative insensitivity to the number of resources, for the problem instances in the test set. Clearly, the growth in the size of the problem deeply impacts the performance of the model due to the combinatorial explosion of the number of binary variables. As in many practical settings, the effectiveness of the model is contingent upon the accuracy of the input data, especially in the demand forecasts, travel times, and cost parameters. Furthermore, the static nature of the model presents limitations when adapted to dynamic real-world operations, which might include disruptions in the demand/supply during the planning horizon. In such cases, our static model must be resolved by fixing some variables to represent the state of the system at the time of the disruption.

Next, we report the results obtained with both the mixed-integer programming formulation and the GRASP-VND matheuristic.

5.2. Final experiments

The goal was to solve the original problem and provide insights for solving other cases in both directions, some that are more complex than the original data and some that are simpler. The instances were designed to analyze the performance of the MIP model, determine its limits, and identify the situations when the matheuristic becomes the better solution approach.

The main goals of these experiments are:

1. to test the limits of the mixed-integer programming (MIP) model and to determine the circumstances under which switching from the proposed MIP to the matheuristic is the better approach for solving the short sea inventory routing problem;
2. to assess the quality of the solutions found with the matheuristic for the instances where the MIP formulation can provide a benchmark.

To test the limits of the MIP, we sequentially solved problem with increased number of ports, periods in the planning horizon, and demand factor. For each instance, we recorded whether at least one integer solution was found, the total computing time, the time to find the best solution, and the optimality gap. When the MIP solver was able to find an integer solution within the first hour of computational time, we stopped the process at the one-hour limit. Otherwise, we continued, for up to 24h, until the first integer solution was found.

Table 3 shows the main results of this experiment. The first three columns identify the group of problem instances. The "Instances with solution" show the number of instances, out of 10, or which the MIP was able to find at least one integer solution. The "Total time" column indicates the average time at the end of the run. The "Time to best" shows the average tie to reach the best solution. The "Gap" is the average optimality gap for the instances with at least on integer solution. The table does not include results for instances with 6 ports and 25 periods because those turned out to be trivial for the MIP solver. Likewise, the MIP solver can provide optimal solutions in a few seconds for instances with 6 ports, 50 periods and demand factor of 1, and therefore those results are also omitted. All other cases that are not shown in Table 3 are due to the inability of the MIP solver to find at least one integer solution to the instances in the corresponding set. This includes all instances with 9 ports and 75 periods and all instances with 12 ports and 50 or 75 periods. For the practical setting that we studied, it seems reasonable to conclude that our MIP model is ideal for problems with 6 ports and no more than 50 periods. It also provides reasonable solutions to all instances with up to 25 periods in the planning horizon. This is consistent with the preliminary analysis shown on Fig. 3.

Table 3
Results of tests with MIP solver.

Ports	Horizon	Demand factor	Instances with solution	Total time	Time to best	Gap
6	50	1.15	5	3600	1462	34.8%
6	75	1	8	3600	3365	47.8%
6	75	1.15	8	5328	4841	47.8%
9	25	1	8	3600	1306	30.4%
9	25	1.15	1	3600	564	34.0%
9	50	1	8	9845	9845	59.9%
9	50	1.15	1	7689	7689	53.0%
12	25	1	6	3600	1961	36.4%
12	25	1.15	3	3600	2093	47.3%

To gain insight on the performance of the matheuristic, we applied it to the problems in Table 3 for which the MIP solver reaches and confirms optimality. The results indicated that the matheuristic was able to find feasible solutions for all the instances while producing an average optimality gap of 11.3% after 30 min and 9.7% after an hour of computational time. Clearly, for the problem sizes in Table 3 for which the MIP solver finds and confirms optimality, this should be the preferred solution method.

We now use all the solutions found with the MIP solver that we summarized in Table 3 (i.e., those solution that are confirmed to be optimal as well as those that are feasible but not confirmed to be optimal) to assess when the matheuristic becomes the better solution alterative. We recorded the solutions found with the MIP solver and the matheuristic after 30 min of computational time and then after one hour. For each instance that the MIP solver can find at least one integer solution, we calculate the deviation between the matheuristic solutions and those found with the MIP model, where negative deviations indicate that the matheuristic found better solutions than the MIP solver.

Table 4 shows the deviations of the matheuristic solutions relative to the MIP solutions. For the 30-minute mark, Table 4 shows the average deviation on column 4. For the 1-hour mark, the table shows minimum, maximum, and average values. The matheuristic was able to find at least one integer solution to all the problems in our data set. We do not show the results for groups without the MIP benchmark, namely, those with 9 ports and 75 periods, as well as 12 ports and 50 or 75 periods. From the results in Table 4, we can conclude that the MIP model is the better option for all problems with a planning horizon of 25 periods and those with 6 ports and up to 50 planning periods. For all other instances, the matheuristic becomes the preferred or in many cases the only option.

In our analysis, we have identified that the matheuristic main difficulty stems from the selection of resources. Throughout our experimentation, we have determined that our "Substitute" move is not enough to produce significant saving in the fixed cost. Since the number and type of resources is relatively small, our recommendation to the company associated with this project was to execute the matheuristic with a preselected set of vessels. In this way, the optimization process focuses on the minimization of the operational costs. To verify that the

Table 4
Deviation values of matheuristic solutions relative to the MIP results.

Ports	Horizon	Demand factor	Avg. deviation (30 min)	Min. deviation (1 h)	Max. deviation (1 h)	Avg. deviation (1 h)
6	50	1.15	20.4%	−7.1%	26.71%	9.95%
6	75	1	−3.3%	−17.7%	4.0%	−3.8%
6	75	1.15	−12.5%	−37.5%	−1.6%	−15.6%
9	25	1	27.0%	17.0%	31.4%	17.0%
9	25	1.15	31.8%	31.3%	31.3%	31.3%
9	50	1	−10.1%	−21.0%	−5.7%	−14.9%
9	50	1.15	−10.6%	−10.6%	−10.6%	−10.6%
12	25	1	36.8%	18.1%	46.8%	35.49%
12	25	1.15	9.8%	−12.9%	24.3%	8.4%

matheuristic performs at a higher level when the resources have been preselected, we reran the procedure on all the problems in Table 4 to calculate a new set of deviation values. These runs were performed using the set of resources chosen by the MIP solutions. We have been able to verify that when the resources are preselected, the matheuristc is the better alternative for all but those instances in which the MIP can reaches optimality (i.e., 6 ports and 25 periods as well as 6 ports and 50 periods with demand factor of 1). The matheuristic is able to reduce the operational cost by 12.9% across the board.

6. Conclusions

Our research addresses a tangible challenge faced by a company in the oil industry, contributing to the operations research literature by introducing a Mixed-Integer Programming model to optimize the transportation of refined products. This model accommodates the complexities of multiple products, a diverse fleet of ships, and long planning horizons, effectively balancing cost components to derive cost-efficient plans.

Through experiments, focusing on a Gulf of Mexico's short sea inventory routing problem, we observed the ability of the MIP model to perform optimally within a 25-day planning horizon or with a limited number of ports. For larger scenarios, a matheuristic approach, combining heuristic search and a simplified mathematical programming model, showed promise in overcoming computational hurdles.

Key findings highlight the substantial cost savings achievable by preselecting resources, particularly relevant as the company often leases rather than own these assets within specific time frames. Despite these advancements, our study recognizes several limitations inherent in the MIP model, including its computational intensity for large instances, reliance on accurate input data, and its static nature without accounting for dynamic real-world operations.

From a managerial point of view, we asked the company to evaluate one of our solutions and the feedback obtained was extremely positive. Specifically, they reported a 6% reduction in total cost, a 5% decrease in assignment cost, a 4% reduction in loading cost, and a 6% decrease in unloading cost. The cost per barrel was estimated to decrease by 6%, and the number of voyages to decrease by 12%. These results underscore the importance of using optimization tools in logistics and operations according to the company's own evaluation of their practices and the proposed solutions in a real scenario.

Future research directions should target the improvement of the matheuristic by exploring additional search strategies that can deal with larger problem instances and integrating stochastic elements to effectively handle uncertainties in real scenarios.

CRediT authorship contribution statement

Sergio Cavero: Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Manuel Laguna:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Eduardo G. Pardo:** Conceptualization, Funding acquisition, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

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

Data availability

The data that has been used is confidential.

Acknowledgements

This research has been partially supported by grants: PID2021-125709OA-C22, FPU19/0409, and EST22/00444 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe"; grant P2018/TCS-4566, funded by Comunidad de Madrid and ERDF; grant CIAICO/2021/224 funded by Generalitat Valenciana; and grant M2988 funded by Proyectos Impulso de la Universidad Rey Juan Carlos 2022; and "Cátedra de Innovación y Digitalización Empresarial entre Universidad Rey Juan Carlos y Second Episode" (Ref. ID MCA06).

References

Agra, A., Andersson, H., Christiansen, M., & Wolsey, L. (2013). A maritime inventory routing problem: Discrete time formulations and valid inequalities. *Networks, 62*, 297–314.
Agra, A., Christiansen, M., & Delgado, A. (2013). Mixed integer formulations for a short sea fuel oil distribution problem. *Transportation Science, 47*, 108–124.
Agra, A., Christiansen, M., & Delgado, A. (2017). Discrete time and continuous time formulations for a short sea inventory routing problem. *Optimization and Engineering, 18*, 269–297.
Agra, A., Christiansen, M., Delgado, A., & Simonetti, L. (2014). Hybrid heuristics for a maritime short sea inventory routing problem. *European Journal of Operational Research, 236*, 924–935.
Christiansen, M. (1999). Decomposition of a combined inventory and time constrained ship routing problem. *Transportation Science, 38*(1), 3–16.
Christiansen, M., Fagerholt, K., & Ronen, D. (2004). Ship routing and scheduling: Status and perspectives. *Trasnportation Science, 38*(1), 1–18.
Christiansen, M., Fagerholt, K., Nygreen, B., & Ronen, D. (2007). Maritime transportation. In C. Barnhart, & G. Laporte (Eds.), *Handbooks in operations research and management science* (pp. 189–284). Amsterdam: North-Holland.
Christiansen, M., Fagerholt, K., Nygreen, B., & Ronen, D. (2013). Ship routing and scheduling in the new millennium. *European Journal of Operational Research, 228*(3), 467–483.
Diz, G. S., Hamacher, S., & Oliveira, F. (2021). A robust optimization model for the maritime inventory routing problem. *Computers & Operations, Research(127)*, Article 105238.
Diz, G. S., Oliveira, F., & Hamacher, S. (2017). Improving maritime inventory routing: Application to a Brazilian petroleum case. *Maritime Policy & Management*, 42–61.
Duarte, A., Sánchez-Oro, J., Mladenović, N., & Todosijević, R. (2018). Variable neighborhood descent. In R. Martí, P. Pardalos, & M. Resende (Eds.), *Handbook of heuristics* (pp. 341–367). Cham: Springer.
Eide, L., Håhjem Årdal, G. C., Evsikova, N., Hvattum, L. M., & Urrutia, S. (2020). Load-dependent speed optimization in maritime inventory routing. *Computers & Operations Research, 105051*.
Engineer, F. G., Furman, K. C., Nemhauser, G. L., Savelsbergh, M. W., & Song, J.-H. (2012). A branch-price-and-cut algorithm for single-product maritime inventory routing. *Operations Research, 60*(1), 106–122.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization, 6*(2), 109–133.

Grønhaug, R., & Christiansen, M. (2009). Supply chain optimization for a liquefied natural gas business. In L. Bertazzi, J. van Nunen, & M. Speranza (Eds.), *Innovations in distribution logistics* (pp. 195–218). Berlin: Springer.

Grønhaug, R., Christiansen, M., Desaulniers, G., & Desrosiers, J. (2010). A branch-and-price method for a liquefied natural gas inventory routing problem. *Transportation Science, 44*(3), 400–415.

Hiassat, A., Diabat, A., & Rahwan, I. (2017). A genetic algorithm approach for location-inventory-routing problem with perishable products. *Journal of Manufacturing Systems*, 93–103.

López-Ibáñez, M.-D.-L. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.

Ronconi, D. P., & Manguino, J. L. (2022). GRASP and VNS approaches for a vehicle routing problem with step cost functions. *Annals of Operations Research*, 1–22.

Ronen, D. (2002). Marine inventory routing: Shipments planning. *Journal of the Operational Research Society, 53*, 108–114.

Sanghikian, N., Martinelli, R., & Abu-Marrul, A. V. (2021). A hybrid VNS for the multi-product maritime inventory routing problem. *International Conference on Variable Neighborhood Search*. Springer International Publishing.

Sherali, H., Al-Yakoob, S., & Hassan, M. (1999). Fleet management models and algorithms for an oiltanker routing and scheduling problem. *IIE Transactions, 31*(5), 395–406.

Siswanto, N., Essam, D., & Sarker, R. (2011). Solving the ship inventory routing and scheduling problem with undedicated compartments. *Computers & Industrial Engineering, 61*(2), 289–299.

Stanzani, A., Pureza, V., Morabito, R., Silva, B., Yamashita, D., & Ribas, P. (2018). Optimizing multiship routing and scheduling with constraints on inventory levels in a Brazilian oil company. *International Transactions in Operational Research, 25*, 1163–1198.

Wang, Y., Nuo, W., & Han, P. (2023). Maritime location inventory routing problem for island supply chain network under periodic freight demand. *Computers & Operations Research, 106042*.