Solving the Liner Shipping Fleet Repositioning Problem with Cargo Flows

Tierney, Kevin; Askelsdottir, Björg; Jensen, Rune Møller; Pisinger, David

Published in:
Transportation Science

Link to article, DOI:
10.1287/trsc.2013.0515

Publication date:
2015

Document Version
Peer reviewed version

TRANSPORTATION SCIENCE
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0041-1655 | EISSN 1526-5447 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

Solving The Liner Shipping Fleet Repositioning Problem with Cargo Flows

Kevin Tierney
IT University of Copenhagen, 2300 Copenhagen S, Denmark, kevt@itu.dk

Björg Áskelsdóttir
DTU Management Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark, bjorgaskels@gmail.com

Rune Møller Jensen
IT University of Copenhagen, 2300 Copenhagen S, Denmark, rmj@itu.dk

David Pisinger
DTU Management Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark, pisinger@man.dtu.dk

We solve a central problem in the liner shipping industry called the Liner Shipping Fleet Repositioning Problem (LSFRP). The LSFRP poses a large financial burden on liner shipping firms. During repositioning, vessels are moved between routes in a liner shipping network. Liner carriers wish to reposition vessels as cheaply as possible without disrupting the cargo flows of the network. The LSFRP is characterized by chains of interacting activities with a multi-commodity flow over paths defined by the activities chosen. Despite its industrial importance, the LSFRP has received little attention in the literature. We introduce a novel mathematical model and a simulated annealing algorithm for the LSFRP with cargo flows that makes use of a carefully constructed graph and evaluate these approaches on real world data from our industrial collaborator. Additionally, we compare the performance of our approach against an actual repositioning scenario, one of many undertaken by our industrial collaborator in 2011. Our simulated annealing algorithm is able to increase the profit earned on our industrial collaborator's scenario from \$18.1 million to \$31.8 million dollars using only a few minutes of CPU time, showing that our algorithm could be used in a decision support system to solve the LSFRP.

Key words: liner shipping, fleet repositioning, maritime optimization

1. Introduction

Responsible for transporting over 1.5 billion tons of cargo in 2012, according to UNCTAD (2012), liner shipping networks reliably and cheaply connect the world's markets. Liner shipping differentiates itself from other forms of shipping due to its fixed, periodic schedules that determine the routes

of vessels, and for the standardized form of cargo, the ISO container, that makes up the majority of the goods carried. This stands in contrast to other forms of shipping, such as tramp or industrial shipping, in which ships generally do not have fixed schedules (Ronen (1983), Christiansen et al. (2004)).

In order for liner shippers to adjust to the needs of their customers, vessels are regularly repositioned between services in liner shipping networks. These repositionings adjust the networks to the world economy and help them to stay competitive. Since repositioning a single vessel can cost hundreds of thousands of US dollars, optimizing the repositioning activities of vessels is an important problem for the liner shipping industry.

The Liner Shipping Fleet Repositioning Problem (LSFRP) consists of finding sequences of activities that move vessels between services in a liner shipping network while respecting the cargo flows of the network. The LSFRP maximizes the profit earned on the subset of the network affected by the repositioning, balancing sailing costs and port fees against cargo and empty equipment revenues, while respecting important liner shipping specific constraints dictating the creation of services and movement of cargo. The subset of the network used in our model is partially defined by the repositioning coordinator in order to help avoid side effects of repositioning from reducing the profitability of the network. A unique feature of the LSFRP is the state-based nature of the activities in the problem. Many LSFRP activities span multiple physical locations and depend on the location of vessels in order to be performed. Automated planning techniques were used to represent a high-level version of the LSFRP that ignored cargo flows in Tierney et al. (2012). Cargo flows, however, are an important aspect of the LSFRP that drive decisions on how vessels should be repositioned.

In this paper, we present a novel mathematical model of the LSFRP with cargo flows on top of a detailed graph that embeds many LSFRP constraints. We solve our model using CPLEX (IBM (2012)) and with a simulated annealing (SA) approach, and study the performance of our model on real world data from our industrial collaborator. We investigate the scaling performance of our model on an actual repositioning scenario in addition to several constructed scenarios. We provide an overview of our parameter tuning procedures for the SA, as well as a comparison of the SA to a reference solution for our actual repositioning instance. This instance models the decisions available to repositioning coordinators as they planned an actual repositioning at Maersk Line involving 11 vessels in 2011. On this instance, our SA is able to find a solution with a profit between $3 million and $13 million higher than the reference solution in only a couple of minutes, thereby doubling the profit earned in the scenario. Our simulated annealing approach is often able to find the optimal solution or very close to the optimal solution and quickly finds solutions for instances that are too large for CPLEX to solve.

It takes a repositioning coordinator several days to find a solution to such a repositioning scenario by hand. Coordinators create a repositioning plan through a trial and error approach which relies on the experience and deep domain knowledge of the coordinator to find a good plan. The coordinators use spreadsheet-like tools to assist them in creating a plan, however they have no optimization capabilities.

We seek to provide an algorithm capable of functioning within a decision support system (DSS) for repositioning coordinators. Within a DSS, the user requires quick answers for planning ship routes. In some cases, the user may need to give feedback to the algorithm, such as not allowing a particular port call or adding extra buffer time on a sailing, so that the plan is real-world feasible. We therefore seek to solve LSFRP instances within a ten minute window.

This paper is structured as follows. We first present a detailed description of the LSFRP in Section 2, including a overview of related work in Section 2.2. Section 3 contains our mathematical model of the LSFRP and graph description. Our SA approach is described in Section 4, followed by a description of our benchmark and a computational evaluation in Section 5. A comparison to our industrial collaborator's repositioning scenario is presented in Section 5.6. Finally, we conclude in Section 6 and present directions for future work. Parts of this paper appeared as an extended abstract in Tierney and Jensen (2012).

2. Liner Shipping Fleet Repositioning

2.1. Problem Description

Liner shipping networks consist of a set of cyclical routes, called services, that visit ports on a regular, periodic schedule[1]. Liner shipping networks are designed to serve customers' cargo demands, which have seasonal fluctuations and shift over time with the world economy. In order to stay competitive, liner carriers must adapt to these changing demands and adjust their networks accordingly. Liner carriers do this by adding, removing and modifying services in their network. Whenever a new service is created, or an existing service is expanded, vessels must be *repositioned* from their current service to the service being added or expanded. Vessel repositioning is expensive due to the cost of fuel (in the region of hundreds of thousands of dollars) and the revenue lost due to cargo flow disruptions. Given that liner carriers around the world reposition hundreds of vessels per year, optimizing vessel movements can significantly reduce the economic and environmental impacts of containerized shipping, and allow carriers to better utilize repositioning vessels to transport cargo.

The aim of the LSFRP is to maximize the profit earned when repositioning a number of vessels from their initial services to a service being added or expanded, called the goal service. We focus

[1] We focus on weekly schedules, as the majority of the services at our industrial collaborator have this structure. However, our work is applicable to any periodic schedule.

on the case where a new service is being added to the network because expanding a service can be seen as a special case of adding a new service, in which vessels are repositioned from the service being expanded to itself along with extra vessels from elsewhere in the network.

Liner shipping services are composed of multiple *slots*, each of which represents a cycle that is assigned to a particular vessel. Each slot is composed of a number of *visits*, which can be thought of as port calls, i.e. a specific time when a vessel is scheduled to arrive at a port. A vessel that is assigned to a particular slot sequentially sails to each visit in the slot. Figure 1 shows a schedule of an example service that contains three slots and visits five ports. The service requires three weeks to complete a cycle, and therefore needs three vessels in order to maintain weekly frequency. Each line (solid black, dashed green, and dotted blue) represents a slot, and each solid circle is a visit at a port at a particular time.

Vessel sailing speeds can be adjusted throughout repositioning to balance cost savings with punctuality. The bunker fuel consumption of vessels increases approximately cubically with the speed of the vessel (Wang and Meng (2012)). *Slow steaming*, in which vessels sail near or at their minimum speed, therefore, allows vessels to sail more cheaply between two ports than at higher speeds, albeit with a longer duration (see, Jorgensen (2011), Meyer et al. (2012)). We linearize the bunker consumption of each repositioning vessel in order to more easily model the LSFRP.

2.1.1. Phase-out & Phase-in The repositioning period for each vessel starts at a specific time when the vessel may cease normal operations, that is, it may stop sailing to scheduled visits and go somewhere else. Each vessel is assigned a different time when it may begin its repositioning, or *phase-out* time. After this time, the vessel may undertake a number of different activities to reach its goal service at low cost. In order to complete the repositioning, each vessel must *phase in* to a slot on the goal service before a time set by the repositioning coordinator. After this time, normal operations on the goal service are set to begin, and all scheduled visits on the service are to be undertaken. In other words, the repositioning of each vessel and optimization of its activities takes place in the period between two fixed times, the vessel's earliest phase-out time and the latest phase-in time of all vessels.

See again Figure 1, which also shows a phase-in service with a phase-in deadline at port c in week 2. After the hollow black circles at port c in weeks 2 through 4, the repositioning vessels must begin regular service, i.e., all visits must be undertaken. Before this time point, visits are only carried out if they are profitable.

Within a *trade zone*, which is a country or group of countries with trade agreements such as the EU or China, vessels may sail freely from their initial service to goal service, as well as back from the goal service to the initial service. However, if two ports lie in different trade zones, vessels

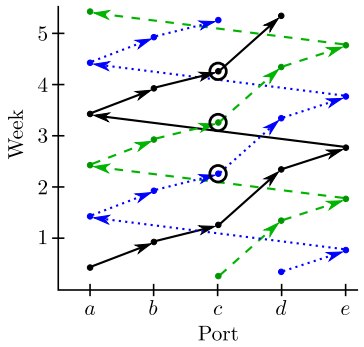

Figure 1 **A time-space graph of a service with three vessels, with a latest phase-in requirement of port c in week 3.**

may only sail between them when going from the initial service to the goal service. We use trade zones to model restrictions preventing cargo from being brought somewhere it might violate the law, or violate a customer's contract on where the shipment may travel. Many countries have laws called *cabotage restrictions* that prevent foreign flagged vessels from offering domestic cargo services. Cabotage restrictions are taken into account during network design, but when vessels are repositioned, their altered paths could result in a violation. Additionally, cargo from certain customers may not be carried on ships that visit certain countries (such as military cargo). We can avoid a detailed modeling of such laws by simply restricting vessels to not leave trade zones when performing certain sailings, such as sailing from visits on the phase-in service to visits on the vessel's initial service.

When a port is visited that is not in the initial or goal service, or is visited out of order, it is called an *inducement*. If a port on the initial or goal service is left off of the repositioning vessel's schedule, it is called an *omission*. Figure 2 shows a vessel's repositioning (solid line) from its initial service (dashed) to its goal service (dotted) within a trade zone. Although Felixstowe is on both the goal and initial services, it is omitted from the repositioning. Note also that the ports Rotterdam and Bremerhaven are induced onto the repositioning path. This is only possible because the induced ports are in the same trade zone as Le Havre and Aarhus (the European Union trade zone).

2.1.2. Cargo and Equipment Revenue is earned through delivering cargo and empty equipment (typically empty containers). We use a detailed view of cargo flows. Cargo is represented as a set of port to port demands with a cargo type, a latest delivery time, an amount of TEU[2] available, and a revenue per TEU delivered. We subtract the cost of loading and unloading each TEU from the revenue to determine the profit per TEU of a particular cargo demand. In contrast to cargo,

[2] TEU stands for *twenty-foot equivalent unit* and represents a single twenty-foot container.

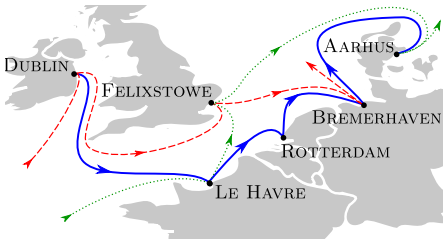

Figure 2 **An example repositioning (blue) from a vessel's phase-out service (dashed red) to its phase-in service (dotted green).**

which can be seen as a multi-commodity flow where each demand is a commodity with a start and end port, empty equipment can be sent from any port where it is in surplus to any port where it is in demand. Ports which have an empty equipment surplus or deficit can be considered to have an infinite amount of supply or demand for a particular type of empty equipment. This is reasonable since the amount of extra containers on-hand or that are required tends to be much greater than the size of a vessel. Each piece of empty equipment brought from a port where it is in excess to a port where it is needed earns a small revenue. The revenue earned is an estimation of how much money was saved by bringing the empty equipment on a repositioning vessel instead of moving the empty equipment through other, more expensive, means.

We consider both *dry* and *reefer* (refrigerated) cargo. Dry containers are standard containers with no specific handling requirements. Reefer containers, in contrast, must be stowed on a vessel in a location with a plug in order to keep the refrigeration unit running. There are, therefore, different capacities on a vessel for dry and reefer containers. Note that although empty equipment can consist of both dry and reefer containers, reefer equipment does not require a reefer slot on a vessel, as containers without any cargo do not need any power source. It is still important to differentiate between dry and reefer equipment, however, as they are not interchangeable for customers.

Some ports have empty equipment, but are not on any service visited by repositioning vessels. These ports are called *flexible* ports, and are associated with flexible visits. The repositioning coordinator may choose the time a vessel arrives at such visits, if at all. All other visits are called *inflexible*, because the time a vessel arrives is fixed.

2.1.3. Sail-on-Service (SoS) Opportunities While repositioning, vessels may use certain services to cheaply sail between two parts of the network. These are called *SoS opportunities*. There are two vessels involved in SoS opportunities, referred to as the *repositioning vessel*, which is the vessel under the control of a repositioning coordinator, and the *on-service vessel*, which is the vessel assigned to a slot on the service being offered as an SoS opportunity. Repositioning vessels can use SoS opportunities by replacing the on-service vessel and sailing in its place for a portion of

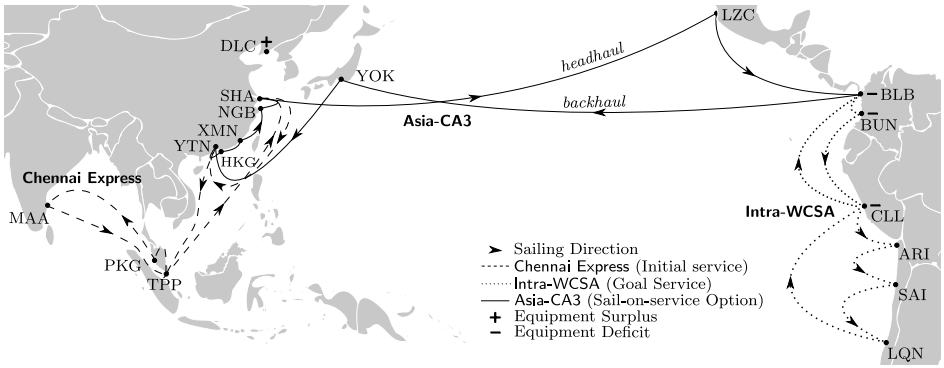

Figure 3 A subset of the case study we performed with our industrial collaborator.

the service. SoS opportunities save significant amounts of money on bunker fuel, since one vessel is sailing where there would have otherwise been two. Using an SoS can even earn money from the *time-charter bonus*, which is money earned by the liner carrier if the on-service vessel is leased.

Consider Figure 3, in which the Asia-CA3 service is offered as a SoS opportunity to the vessel repositioning from Chennai Express to Intra-WCSA. The repositioning vessel can leave the Chennai Express at TPP, and sail to HKG where it picks up the Asia-CA3, replacing the on-service vessel. The repositioning vessel then sails along the Asia-CA3 service until it gets to BLB where it can join the Intra-WCSA. Note that no vessel sails on the backhaul of the Asia-CA3, and this is allowed because very little cargo travels on the Asia-CA3 towards Asia.

When a repositioning vessel uses an SoS opportunity, the on-service vessel is either laid-up or leased out, freeing a slot on the service. The repositioning vessel may join the freed slot in any of the *starting visits* and may leave the slot in one of the *ending visits*. There are two ways for repositioning vessels to start an SoS: *transshipment* and *parallel sailing*. When starting an SoS by transshipment, all cargo loaded on the on-service vessel is transshipped (moved) to the repositioning vessel. Each TEU transshipped has a fee roughly equal the cost of loading a TEU. Transshipment is not always possible due to the previously described cabotage restrictions that exist in some countries, or prohibitively expensive. Illegal or expensive transshipments can be avoided using a *parallel sailing*, in which both the repositioning vessel and the on-service vessel visit ports in tandem. The repositioning vessel only loads cargo, and the on-service vessel only discharges cargo. Since two vessels are sailing in tandem during a parallel sailing, fuel consumption is doubled, as are port fees. However, this is still sometimes cheaper than transshipping cargo directly between vessels.

2.2. Literature Review

The LSFRP has received little attention in the literature and was not mentioned in any of the most influential surveys of work in the liner shipping domain (Christiansen et al. (2013, 2007, 2004))

or container terminals (Steenken et al. (2004), Stahlbock and Voß (2008)). Although there has been significant work on problems such as the Fleet Deployment Problem (FDP) (e.g., Powell and Perakis (1997)) and the Network Design Problem (NDP) (e.g., Agarwal and Ergun (2008), Álvarez (2009), Brouer et al. (2012)), these problems deal with strategic decisions related to building the network and assigning vessels to services, rather than the operational problem of finding paths for vessels through the network.

Although tramp shipping problems, such as in Christiansen (1999) and Korsvik et al. (2011), maximize cargo profit while accounting for sailing costs and port fees as in the LSFRP, they lack liner shipping specific constraints, such as SoS opportunities, phase-in requirements and strict visit times. Airline disruption management (see Kohl et al. (2007), Clausen et al. (2010)), while also relying on time-based graphs, differs from the LSFRP in two key ways. First, airline disruption management requires an exact cover of all flight legs over a planning horizon. The LSFRP has no such requirement over visits or sailing legs. Second, there are no flexible visits in airline disruption management.

The vessel schedule recovery problem (VSRP), see Brouer et al. (2013), focuses on recovering operations after a disruption, such as bad weather or mechanical failure, delays a container vessel. Similar to the LSFRP, the VSRP must respect the periodic frequency of services and network cargo flows. However, the two problems differ in that the VSRP lacks many cost saving aspects of the LSFRP because it is solved over a much shorter time window with only a single vessel.

Andersen (2010) discusses a problem similar to the LSFRP, called the Network Transition Problem (NTP). No mathematical model or formal problem description is provided, so it is difficult to exactly ascertain what the NTP solves in comparison to the LSFRP. However, it is clear that the NTP lacks cost saving activities like SoS opportunities, empty equipment flows and slow steaming.

The primary previous work on the LSFRP in the literature is found in Tierney et al. (2012), Kelareva et al. (2013) and Tierney and Jensen (2012). The first work on the LSFRP, Tierney et al. (2012), solved an abstraction of the LSFRP without cargo or empty equipment flows and SoS parallel sailings using Linear Temporal Optimization Planning (LTOP), a hybrid of automated planning and linear programming that performs a branch-and-bound search for repositioning solutions. However, LTOP and other automated planning methods are unable to model cargo flows and are thus inapplicable to the LSFRP with cargo flows. In Kelareva et al. (2013), a constraint programming approach is presented for the previously mentioned repositioning problem.

A mathematical model of the LSFRP with cargo and equipment flows is introduced in Tierney and Jensen (2012), and CPLEX is used to solve the model. While CPLEX is able to solve a number of instances to optimality, many instances are too large for CPLEX to tackle. This paper extends that work with some model improvements, a public dataset, and a heuristic approach that solves the instances that are too big for CPLEX to solve.

3. Mathematical Model

We model the LSFRP with cargo flows on a graph $G = (V, A)$, where V is the set of nodes and A the set of directed arcs between nodes. Each node in V represents a visit of a vessel at a particular port[3], and each arc in A represents an allowed sailing between two visits. The graph encompasses all of the activities each vessel may undertake during a fixed *repositioning period*, which is the period from the time the vessel is first allowed to leave its phase-out service until the time when normal operations must begin on the phase-in service. The path of each vessel through the graph represents the activities to be undertaken by that vessel, and we therefore require the paths to be node disjoint to prevent multiple vessels from performing the same activity. This is an important constraint because *i*) container port terminals assign timeslots to vessels, meaning there is not enough room for two vessels, and *ii*) profit from carrying cargo can only be earned a single time, removing any reason for multiple vessels to visit the same node. Note that flexible visits, i.e. visits without a prior fixed schedule, can be undertaken by multiple vessels, even simultaneously. For ease of modeling, we therefore replicate flexible visits for each vessel and consider them as node disjoint. We give more details about this process (and its justifications) later. We embed a number of problem constraints and objectives directly in the graph, including sailing costs, sail-on-service opportunities, cabotage restrictions, phase-in/out requirements, and canal fees, which are described in detail in the next section, followed by our MIP model over the graph.

3.1. Graph Description

We give a textual overview of the graph used in our model of the LSFRP with cargo flows. For a detailed formal description, we refer the reader to Appendix A. The visits in the graph are split into two sets, thus $V = V^i \cup V^f$, where V^i is the set of *inflexible* visits, i.e. visits associated with a specific port call time, and V^f is the set of *flexible* visits, which are assigned a time only if a vessel performs the visit. The set V^f contains visits in which a vessel can pick up/deliver equipment or incremental cargo that are not on any phase-out, phase-in, or SoS service. Let S be the set of ships.

Visits are connected with either inflexible or flexible arcs represented by the sets A^i and A^f, respectively. Inflexible arcs have fixed durations that can be pre-computed, whereas the time a ship is sailing on a flexible arc must be determined during optimization.

The overall structure of the graph involves four types of visits: phase-out, phase-in, flexible, and SoS visits. In addition to these visits, we include a graph sink, τ, which all vessels must reach for a valid repositioning. We let $V' = V \setminus \tau$ be the set of all graph visits excluding τ. The four types of visits represent four disjoint sets that make up V'. We now describe the arc structure corresponding to each of the four types of visits.

[3] We use the terms visit and node interchangeably.

3.1.1. Phase-out Each ship is assigned a particular visit, $\sigma_s \in V'$, at which the ship $s \in S$ begins its repositioning. This visit represents the earliest allowed phase-out time for that vessel. A visit is then created for each subsequent port call of the ship on its phase-out slot. Each phase-out visit is connected to the next one with an arc, as well as to all subsequent phase-in visits. Note that phase-out visits do not connect to the phase-out visits of other ships.

Vessels may leave phase-out nodes to sail to SoS opportunities, flexible nodes, or to a phase-in slot. Thus, arcs are created from each phase-out visit to each phase-in visit and SoS start visit such that sailing between the visits is temporally feasible (i.e. the starting time of the phase-in/SoS visit is greater than the end time of the phase-out visit plus the sailing time). Since flexible nodes have no fixed start and end time, arcs are created to and from all flexible nodes to all phase-outs within the same trade zone. Finally, phase-out visits have incoming arcs from phase-in visits in the same trade zone. This allows ships to avoid sailing back and forth between ports when transferring directly between the phase-out and phase-in.

3.1.2. Phase-in We create visits for each port call along a phase-in slot, and connect subsequent phase-in visits to each other. The final visit in a slot, which represents the time at which regular operations must begin on a service, is connected to the graph sink, τ.

The phase-in graph structure ensures that the goal service has a vessel in each of its slots. An example phase-in graph structure is portrayed in Figure 4, which shows the graph for the service in Figure 1. Each sequence of visits (from top to bottom: red, green blue) represents a slot on the goal service. Each visit is labeled with the port and week that it is visited. We note that our model does not require services to have a weekly regularity, and can handle any service with a consistent periodicity (i.e., every two weeks or every 15 days). The last node in each sequence corresponds to the on-time requirement (node $(c, 2)$) extended to each slot. After each of these visits, the service begins normal operations, and is no longer under the control of the repositioning coordinator. This graph structure ensures that all vessels perform a legal phase-in, namely that each slot is assigned a single vessel. Each phase-in slot is guaranteed to be assigned a single vessel since there are as many slots as there are vessels (three), the graph sink τ only has a single incoming node from each slot, and the paths of vessels are node disjoint (except for τ).

3.1.3. Flexible visits Flexible visits are modeled by replicating each flexible visit for each ship in the model. Flexible visits are connected to all inflexible and flexible visits within the same trade zone. Replicating flexible visits for each vessel avoids requiring special constraints in the MIP model to handle the fact that multiple vessels can visit the same flexible visit. This is because when a vessel visits a flexible visit, the visit must be assigned a time when it can take place. Simply copying the flexible nodes ensures that each flexible node can be scheduled along the path of a

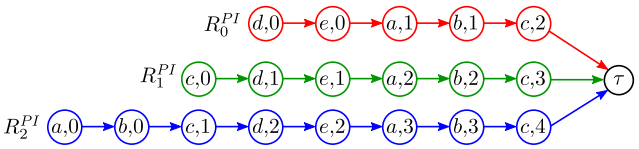

Figure 4 **The phase-in graph structure for the service in Figure 1. The sets R_0^{PI}, R_1^{PI}, and R_2^{PI} contain the nodes for each phase-in slot. Each set of nodes ends with a single visit at port c on weeks 2, 3 and 4, ensuring that the weekly structure of the service is enforced.**

vessel with simple constraints. Since our instances generally do not contain many flexible visits, this duplication does not significantly hinder the solvability of the instances. Note that this opens the possibility that two vessels may visit the same flexible visit at the same time. We do not consider this to be a problem since flexible visits are at ports that will probably have the capacity to deal with multiple ships. Since flexible visits do not have fixed entry and exit times, the time required for a vessel to visit them must be taken into account. The total port stay at a flexible visit consists of the *piloting time*, which is the time required to maneuver the vessel into, and out of, a port, and the cargo/equipment (un)loading time. Flexible visits are connected to other visits in the graph through the set of flexible arcs, A^f.

3.1.4. Sail-on-service We introduce a number of disjoint sets of graph arcs and graph nodes to represent a special graph structure that models SoS opportunities. We view an SoS as having three types of ports; *entry ports*, where vessels may join the SoS, *through ports*, in which a vessel must already be on the SoS, and *end ports* where a vessel may leave the SoS. The designation of these ports is left to the repositioning coordinator. We make this distinction in case there are circumstances outside the scope of the model that require certain ports to be called if an SoS is used.

Figure 5 shows the graph structure of an example SoS opportunity, o. Vessels may enter the SoS using arcs in the set $\hat{A}_o^{In}$, either through parallel sailing nodes (O_o^P) or transshipment nodes (O_o^{TS}), shown in red and green, respectively. Parallel sailings end in a transshipment, in which cargo is moved to the repositioning vessel. The parallel sailing nodes are connected to the transshipment nodes with the set of arcs $\hat{A}_o^{PTS}$. The set of arcs $\hat{A}_o^{PP}$ contains arcs connecting subsequent ports for parallel sailings. These arcs have twice the sailing cost for each vessel as the other arcs in the SoS graph structure, which have the normal sailing cost between two ports for each vessel. Note that p_3 has no parallel sailing node because transshipment is not allowed in p_4, which is a through port. Ports with cabotage restrictions, such as p_1, do not receive transshipment nodes, as transshipping at that particular port would violate the law. Transshipment nodes connect to through nodes (blue) using the arcs $\hat{A}_o^{TST}$. Once at a through node, a vessel must sail onward to the next through node using the arc set $\hat{A}_o^{TT}$, until it reaches the arc $\hat{a}_o^{TE}$. This arc connects the through nodes to the

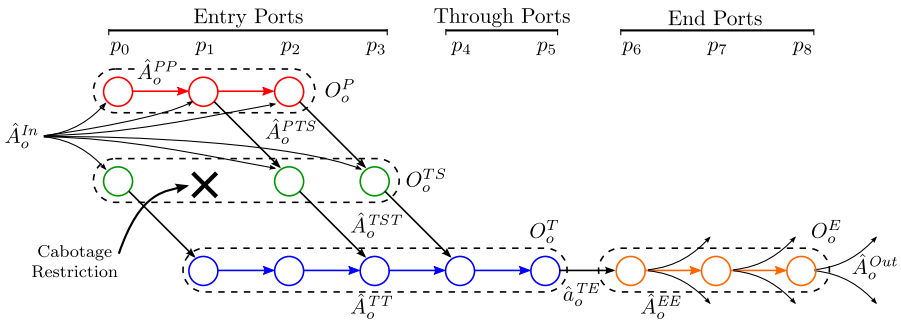

Figure 5 **The graph structure of an example SoS opportunity, which contains parallel nodes O_o^P in red, transshipment nodes O_o^{TS} in green, a cabotage restriction at port p_1, through nodes O_o^T in blue, and end nodes O_o^E in orange.**

end nodes, and represents a bottleneck that ensures only one vessel uses a particular SoS. Since the paths of the vessels are node disjoint, two vessels would have to visit the latest node in O_o^T and the earliest node in O_o^E, which is not allowed. Finally, vessels may exit the SoS through the end nodes (orange) in the set O_o^E, using an arc in the set $\hat{A}_o^{Out}$. End nodes are connected by the arcs in the set $\hat{A}_o^{EE}$.

3.1.5. Sailing Cost The fuel consumption of a ship is approximately a cubic function of the speed of the vessel (Wang and Meng (2012)). We precompute the optimal cost for each inflexible arc using a linearized bunker consumption function, and compute the costs of flexible arcs during optimization in our MIP model. All inflexible arcs in the model are assigned a sailing cost for each ship that is the optimal sailing cost given the total duration of the arc. Since ships have a minimum speed, if the duration of the arc is greater than the time required to sail on the arc at a ship's minimum speed, the cost is calculated using the minimum speed and the ship simply waits for the remainder of the duration. This is a common practice for liner carriers in order to add buffer to their schedules, thus making the network more robust to disruptions.

3.2. MIP Model

We now define the MIP model that guides the vessels through the graph, and controls the flow of cargo and empty equipment, using the following parameters and variables to supplement the parameters used to define the graph.

3.2.1. Parameters

S	Set of ships.
V'	Set of visits minus the graph sink.
V^i, V^f	Set of inflexible and flexible visits, respectively.
A^i, A^f	Set of inflexible and flexible arcs, respectively.
A'	Set of arcs minus those arcs connecting to the graph sink, i.e., $(i,j) \in A$, $i,j \in V'$.

Q	Set of empty equipment types. $Q = \{dc, rf\}$.
M	Set of demand triplets of the form (o, d, q), where $o \in V', d \subseteq V'$ and $q \in Q$ are the origin visit, destination visits and the cargo type, respectively.
$V^{q+} \subseteq V'$	Set of visits with an equipment surplus of type q.
$V^{q-} \subseteq V'$	Set of visits with an equipment deficit of type q.
$V^{q^*} \subseteq V'$	Set of visits with an equipment surplus or deficit of type q ($V^{q^*} = V^{q^+} \cup V^{q^-}$).
$u_s^q \in \mathbb{R}^+$	Capacity of vessel s for cargo type $q \in Q$.
$M_i^{Orig}, (M_i^{Dest}) \subseteq M$	Set of demands with an origin (destination) visit $i \in V$.
$v_s \in V'$	Starting visit of ship $s \in S$.
$t_{si}^{Mv} \in \mathbb{R}$	Move time per TEU for vessel s at visit $i \in V'$.
$t_i^E \in \mathbb{R}$	Enter time at inflexible visit $i \in V'$.
$t_i^X \in \mathbb{R}$	Exit time at inflexible visit $i \in V'$.
$t_i^P \in \mathbb{R}$	Pilot time at visit $i \in V'$.
$r_q^{Var} \in \mathbb{R}^+$	Revenue for each TEU of equipment of type $q \in Q$ delivered.
$r^{(o,d,q)} \in \mathbb{R}^+$	Amount of revenue gained per TEU of loaded containers carried for the demand triplet.
$c_{sij}^{Sail} \in \mathbb{R}^+$	Fixed cost of vessel s utilizing arc $(i, j) \in A'$.
$c_{sij}^{VarSail} \in \mathbb{R}^+$	Variable hourly cost of vessel $s \in S$ utilizing arc $(i, j) \in A'$.
$c_i^{Mv} \in \mathbb{R}^+$	Cost to move a single TEU on or off a ship at visit $i \in V'$.
$c_{si}^{Port} \in \mathbb{R}$	Port fee associated with vessel s at visit $i \in V'$.
$\Delta_{ijs}^{Min} \in \mathbb{R}^+$	Minimum duration for vessel s to sail on flexible arc (i, j).
$\Delta_{ijs}^{Max} \in \mathbb{R}^+$	Maximum duration for vessel s to sail on flexible arc (i, j).
$a^{(o,d,q)} \in \mathbb{R}^+$	Amount of demand available for the demand triplet.
$In(i) \subseteq V'$	Set of visits with an arc connecting to visit $i \in V$.
$Out(i) \subseteq V'$	Set of visits receiving an arc from $i \in V$.
$\tau \in V$	Graph sink, which is not an actual visit.

3.2.2. Variables

$w_{ij}^s \in \mathbb{R}_0^+$	The amount of time that vessel $s \in S$ sails on flexible arc $(i, j) \in A^f$.
$x_{ij}^{(o,d,q)} \in \mathbb{R}_0^+$	Amount of flow of demand triplet $(o, d, q) \in M$ on $(i, j) \in A'$.
$x_{ij}^q \in \mathbb{R}_0^+$	Amount of empty equipment of type $q \in Q$ flowing on $(i, j) \in A'$.
$y_{ij}^s \in \{0, 1\}$	Indicates whether vessel s is sailing on arc $(i, j) \in A$.
$z_i^E \in \mathbb{R}_0^+$	Defines the entrance time of a vessel at visit i.
$z_i^X \in \mathbb{R}_0^+$	Defines the exit time of a vessel at visit i.

3.2.3. Objective and Constraints

$$\max \sum_{(o,d,q) \in M} \left(\sum_{j \in d} \sum_{i \in In(j)} \left(r^{(o,d,q)} - c_o^{Mv} - c_j^{Mv} \right) x_{ij}^{(o,d,q)} \right) \tag{1}$$

$$+ \sum_{q \in Q} \left(\sum_{i \in V^{q+}} \sum_{j \in Out(i)} \left(r_q^{Eqp} - c_i^{Mv} \right) x_{ij}^q - \sum_{i \in V^{q-}} \sum_{j \in In(i)} c_i^{Mv} x_{ji}^q \right) \tag{2}$$

$$- \sum_{s \in S} \left(\sum_{(i,j) \in A'} c_{sij}^{Sail} y_{ij}^s + \sum_{(i,j) \in A^f} c_{sij}^{VarSail} w_{ij}^s \right) \tag{3}$$

$$- \sum_{j \in V'} \sum_{i \in In(j)} \sum_{s \in S} c_{sj}^{Port} y_{ij}^{s} \tag{4}$$

$$\text{s.t.} \sum_{s \in S} \sum_{i \in In(j)} y_{ij}^{s} \leq 1 \qquad \forall j \in V' \tag{5}$$

$$\sum_{j \in Out(i)} y_{ij}^{s} = 1 \qquad \forall s \in S, i = v_s \tag{6}$$

$$\sum_{i \in In(\tau)} \sum_{s \in S} y_{i\tau}^{s} = |S| \tag{7}$$

$$\sum_{i \in In(j)} y_{ij}^{s} - \sum_{i \in Out(j)} y_{ji}^{s} = 0 \qquad \forall j \in \{V' \setminus \bigcup_{s \in S} v_s\}, s \in S \tag{8}$$

$$\sum_{(o,d,rf) \in M} x_{ij}^{(o,d,rf)} \leq \sum_{s \in S} u_s^{rf} y_{ij}^{s} \qquad \forall (i,j) \in A' \tag{9}$$

$$\sum_{(o,d,q) \in M} x_{ij}^{(o,d,q)} + \sum_{q' \in Q} x_{ij}^{q'} \leq \sum_{s \in S} u_s^{dc} y_{ij}^{s} \qquad \forall (i,j) \in A' \tag{10}$$

$$\sum_{i \in Out(o)} x_{oi}^{(o,d,q)} \leq a^{(o,d,q)} \sum_{i \in Out(o)} \sum_{s \in S} y_{oi}^{s} \qquad \forall (o,d,q) \in M \tag{11}$$

$$\sum_{i \in In(j)} x_{ij}^{(o,d,q)} - \sum_{k \in Out(j)} x_{jk}^{(o,d,q)} = 0 \qquad \forall (o,d,q) \in M, j \in V' \setminus (o \cup d) \tag{12}$$

$$\sum_{i \in In(j)} x_{ij}^{q} - \sum_{k \in Out(j)} x_{jk}^{q} = 0 \qquad \forall q \in Q, j \in V' \setminus V^{q^*} \tag{13}$$

$$\Delta_{ijs}^{Min} y_{ij}^{s} \leq w_{ij}^{s} \leq \Delta_{ijs}^{Max} y_{ij}^{s} \qquad \forall (i,j) \in A^f, s \in S \tag{14}$$

$$z_i^E = t_i^E \sum_{s \in S} \sum_{j \in In(i)} y_{ij}^{s} \qquad \forall i \in V^i \tag{15}$$

$$z_i^X = t_i^X \sum_{s \in S} \sum_{j \in Out(i)} y_{ij}^{s} \qquad \forall i \in V^i \tag{16}$$

$$z_i^X + \sum_{s \in S} w_{ij}^{s} \leq z_j^E \qquad \forall (i,j) \in A^f \tag{17}$$

$$\sum_{(o,d,q) \in M_i^{Orig}} \sum_{j \in Out(o)} t_{so}^{Mv} x_{oj}^{(o,d,q)} + \sum_{(o,d,q) \in M_i^{Dest}} \sum_{k \in d} \sum_{j \in In(k)} t_{sd}^{Mv} x_{jk}^{(o,d,q)}$$
$$+ \sum_{q \in Q} \left(\sum_{i' \in \{V^{q+} \cap \{i\}\}} \sum_{j \in Out(i')} t_{si'}^{Mv} x_{i'j}^{q} + \sum_{i' \in \{V^{q-} \cap \{i\}\}} \sum_{j \in In(i')} t_{sj}^{Mv} x_{ji'}^{q} \right)$$
$$- \; z_i^X + z_i^E + t_i^P \sum_{j \in In(i)} y_{ij}^{s} \leq 0 \qquad \forall i \in V^f, s \in S \tag{18}$$

The domains of the variables are as previously described. The objective consists of several components. The profit from delivering cargo (1) is computed based on the revenue from delivering cargo minus the cost to load and unload the cargo from the vessel. Note that the model can choose how much of a demand to deliver, even choosing to deliver a fractional amount. We can allow this

since each demand is an aggregation of cargo between two ports, meaning at most one container between two ports will be fractional. Furthermore, since each individual demand triplet can have multiple possible destinations we sum over all of these destinations, d. Empty equipment profit is taken into account in (2). Empty equipment is handled similar to cargo, except that equipment can flow from any port where it is in supply to any port where it is in demand. The sailing cost (3) takes into account the precomputed sailing costs on arcs between inflexible visits, as well as the variable cost for sailings to and from flexible visits. Note that the fixed sailing cost on an arc does not only include fuel costs, but can also include canal fees or the time-charter bonus for entering an SoS. Finally, port fees are deducted in (4).

Multiple vessels are prevented from visiting the same visit in (5). The flow of each vessel from its source node to the graph sink is handled by (6), (7) and (8), with (7) ensuring that all vessels arrive at the sink.

Arcs are assigned capacities if a vessel utilizes the arc in (9), which assigns the reefer container capacity, and in (10), which assigns the total container capacity, respectively. Note that constraints (9) do not take into account empty reefer equipment, since empty containers do not need to be turned on, and can therefore be placed anywhere on the vessel. Cargo is only allowed to flow on arcs with a vessel in (11). The flow of cargo from its source to its destination, through intermediate nodes, is handled by (12). Constraints (13) balance the flow of empty equipment into and out of nodes. In contrast to the way cargo is handled, equipment can flow from any port where it is in supply to any port where it is in demand. Since the amount of equipment carried is limited only by the capacity of the vessel, no flow source/sink constraints are required.

Flexible arcs have a duration constrained by the minimum and maximum sailing time of the vessel on the arc in (14). The enter and exit time of a vessel at inflexible ports is handled by (15) and (16), and we note that in practice these constraints are only necessary if one of the outgoing arcs from an inflexible visit ends at a flexible visit. Constraints (17) sets the enter time of a visit to be the duration of a vessel on a flexible arc plus the exit time of the vessel at the start of the arc. Constraints (18) controls the amount of time a vessel spends at a flexible visit. The first part of the constraint computes the time required to load and unload cargo and equipment, with the final term of the constraint adding the piloting time to the duration only if one of the incoming arcs is enabled (i.e. the flexible visit is being used).

The model forms a disjoint path problem in which a fractional multi-commodity flow is allowed to flow over arcs in the vessel paths, along with a small scheduling component in the flexible nodes. Flexible arcs could be alternatively represented using a discretized approach, however we forego a discretization because of the vast differences in timescales between port activities and sailing activities, which are on the order of hours and days, respectively. In order to achieve such a fine

grained view of flexible arc activities, we would require numerous extra arcs and nodes for each flexible node.

3.3. Complexity

We reduce the knapsack problem to the LSFRP with flexible visits in order to show that the LSFRP is NP-complete. Given n items, each with a profit p_i and a size s_i, and a knapsack with a capacity C, the knapsack problem maximizes the objective $\sum_{i=0}^{n} p_i x_i$ where x_i is a binary variable indicating whether or not item i is in the knapsack, subject to the capacity constraint $\sum_{i=0}^{n} s_i x_i \leq C$.

THEOREM 1. *The LSFRP with flexible visits is NP-complete.*

We first note that the LSFRP is clearly in NP, as the total profit can be easily computed from the paths of the vessels through the graph. We initialize an LSFRP with a single vessel and no cargo or equipment demands. The problem instance contains a single phase-out visit, ω, and a single phase-in visit, λ. The port fees at both ω and λ are 0, and we let $\mathit{enter}(\omega) = \mathit{exit}(\omega) = 0$ and $\mathit{enter}(\lambda) = \mathit{exit}(\lambda) = C$. In other words, the timespan in which the repositioning must take place is limited to the capacity of the knapsack. For each knapsack item, we create a flexible visit, f_i, which has a duration of exactly s_i, i.e. $\mathit{pilot}(f_i) = s_i$. The port fee for visiting f_i is $-p_i$, since the LSFRP maximizes profit (i.e. minimizes fees). All flexible nodes, as well as ω and λ, are in a single trade zone. Therefore, the specification of the LSFRP graph ensures that the phase-out node, ω, connects to all flexible nodes, all flexible nodes connect to each other, and all flexible nodes connect to the phase-in node, λ. The sailing time of the vessel between all nodes in the graph is set to 0.

Item i is included in the knapsack solution if and only if the vessel visits f_i during its repositioning. Since the vessel can only visit a single flexible visit at a time, the duration of each flexible visit is fixed to the size of the item it represents, and the phase-in visit is fixed in time to the size of the knapsack, the capacity constraint of the knapsack must be satisfied. Additionally, according to the objective of the LSFRP, only the flexible visits corresponding to the maximum profit knapsack items will be chosen. Therefore, the LSFRP with flexible visits is NP-complete. □

We note that flexible ports are not present in every LSFRP problem, and this proof only covers those with flexible ports. This is not to say that LSFRP problems without flexible ports are necessarily polynomial time solvable. Indeed, the LSFRP without flexible ports is likely NP-complete, however this is not trivial to prove and remains an open problem at this time.

4. Simulated Annealing

We created a heuristic solution procedure for the LSFRP with cargo flows using a simulated annealing (SA) algorithm. SA was introduced in Kirkpatrick et al. (1983) and consists of a local search that tries to avoid getting stuck in a local optimum by accepting worsening solutions with

a probability proportional to a so-called *temperature*, which declines as the algorithm progresses. SA therefore begins almost as a random walk, and slowly becomes more greedy as it explores the search space, eventually converging on a local optimum.

We use a combined reheating and restart strategy similar to the one used in Taheri and Zomaya (2007) to overcome local optima. Reheating involves increasing the temperature of the SA after convergence to a factor of the initial temperature. We combine reheating with full restarts, in that we allow only several reheats before we restart the SA from the initial solution. The idea behind such a restart is that several reheats could put the solution of the SA in a part of the search space that is far away from the global optimum, and continual reheating may not move it in the correct direction. Restarting from the initial solution allows the SA to move in a different direction and find a better solution. Our solution procedure can be viewed as a form of iterated local search (see Lourenço et al. (2003)), in which reheating is the perturbation procedure and the local search to be iterated over is simulated annealing. Our SA algorithm uses a penalized objective in which certain types of infeasible solutions are accepted in order to avoid getting stuck within the search landscape.

Algorithm 1 shows the particular version of the SA algorithm that we are using in this work, parameterized as follows: p represents the problem to solve, f is the objective evaluation function, t^{Init} is the initial temperature, α is the temperature reduction factor, β is the reheating factor, t^{Min} is the convergence temperature, r^{Itrs} is the maximum number of non-improving iterations before reheating, $r^{Restart}$ is the number of reheats before resetting the solution to the initial solution, r^{Reheat} is the number of non-improving reheats before stopping the search.

After creating an initial solution on line 2 and initializing variables on the following lines, the algorithm begins its outer reheat/restart loop. On lines 7 – 9 we reset the solution used for the current reheat if the number of reheats exceeds a parameter $r^{Restart}$. This allows the SA to choose a different path from the starting solution, one that could lead to better solutions. Within the SA inner loop on lines 10 – 18, a random neighbor of the current solution is selected.The neighbor selected on line 11 is generated randomly from one of several neighbor generators. We choose the generator uniformly at random. We then check whether the objective of the new solution is better than that of the current solution. When the new solution has a better objective value, we accept it. However, when the new solution is worse than the current solution, we accept the solution according to the Metropolis criterion introduced in Kirkpatrick et al. (1983). We then update the incumbent solution (line 16) and reduce the temperature on an exponential cooling schedule according to the factor α on line 17. We exit the inner loop if the temperature falls below the threshold, t^{Min}, or the number of iterations in which no improving solution was found is greater than r^{Itrs}. We reheat the temperature to a factor β of the original temperature and continue the search until either running out of CPU time or we exceed the maximum number of non-improving reheats, r^{Reheat}.

Algorithm 1 The SA algorithm with reheating and restarts.

0.9

function SA(p, f, t^{Init}, α, β, t^{Min}, r^{Itrs}, $r^{Restart}$, r^{Reheat})
  $s^{Init} \leftarrow$ CreateSolution(p)
  $t \leftarrow t^{Init}$; $s^* \leftarrow s^{Init}$; $s^*_{prev} \leftarrow s^{Init}$
  $reheats \leftarrow 0$; $nonImprovingReheats \leftarrow 0$
  repeat ▷ Reheat/restart loop.
    $nonImprovingItr \leftarrow 0$;
    if $reheats \geq r^{Restart}$ **then**
      $s \leftarrow s^{Init}$
      $reheats \leftarrow 0$
    repeat ▷ SA loop.
      $s' \leftarrow$ SelectNeighbor(s)
      if $f(s') > f(s)$ **then** $s \leftarrow s'$
      else
        $nonImprovingItr \leftarrow nonImprovingItr + 1$
        if $\exp((f(s') - f(s))/t) >$ Random() **then** $s \leftarrow s'$ ▷ Metropolis acceptance criterion.
      if $f(s') > f(s^*)$ **then** $s^* \leftarrow s'$
      $t \leftarrow t\alpha$
    until $t < t^{Min}$ **or** $nonImprovingItr \geq r^{Itrs}$
    $t \leftarrow t^{Init}\beta$ ▷ Reheat to a factor β of the initial temperature.
    $reheats \leftarrow reheats + 1$
    if $s^* \leq s^*_{prev}$ **then** $nonImprovingReheats \leftarrow nonImprovingReheats + 1$
    $s^*_{prev} \leftarrow \max\{s^*, s^*_{prev}\}$
  until *Time limit reached* **or** $nonImprovingReheats \geq r^{Reheat}$
return s^*

4.1. Initial Solution Heuristics

We introduce several heuristics to generate initial solutions to the LSFRP for use in our SA algorithm.

4.1.1. Direct route heuristic (DRH) We model the connections between the starting visit of each vessel with all of the feasible phase-ins using a linear assignment problem. The cost of each vessel/phase-in assignment is equal to the sailing cost of the vessel to the particular phase-in if

the sailing is feasible, or infinity if it is not. The direct route heuristic generates a feasible starting solution, but it is rarely, if ever, an optimal solution to the LSFRP. We use the Hungarian algorithm (Kuhn (1955)) to solve the linear assignment problem.

4.1.2. Shortest paths heuristic (SPH) We generate paths for vessels using a shortest path algorithm that iteratively creates a path for each ship on a graph containing only those visits that are not visited in any previously generated path. While this ensures that any solution the heuristic generates will be feasible, it may not always be possible to generate any solution at all since all of the feasible phase-in opportunities for a particular ship may already have been assigned to another ship. We order the ships by their first possible phase-out time and generate paths starting with the ship with the latest phase-out. We found that with this ordering, only one out of our 44 instances could not generate a feasible starting solution, whereas with random orderings or with an ascending phase-out time ordering, feasible starting solutions could almost never be generated. If it is not possible to generate a solution, we fall back on DRH, although this never happened in our experiments.

Sailing costs on inflexible arcs as well as port fees are easy to take into account in the shortest path algorithm, however flexible arcs and cargo/equipment revenues pose a challenge. It is not possible to take these components fully into account within a standard shortest path algorithm, since this would require the cost of a particular arc to vary based on the scheduling of flexible arcs. Thus, the heuristic generates solutions that do not re-use visits between vessels, but are not necessarily temporally feasible on instances with flexible arcs.

Flexible arc handling We allow flexible arcs to be used in the shortest path, even though they represent a scheduling problem that cannot be solved while the shortest path algorithm is running. We ignore temporal feasibility and focus only on the cost of the flexible arc. We define a parameter γ in the range $[0, 1]$ that represents how fast the ship is sailing with a value of 0 being the ship's minimum speed, and a value of 1 being the maximum speed. We then assign arcs a sailing cost based on the speed of the vessel. This allows the heuristic to try to take into account some of the costs that would be incurred using a flexible arc.

Cargo handling The profit for each cargo demand in the graph is represented by computing the total possible profit from the demand (cargo revenue less container loading/unloading fees at the origin and destination) and multiplying this value by a scaling parameter ℓ^{Cargo}, which is in the range $[0, 1]$. We then offset the sailing costs and port fees at the origin and destination visits using this scaled cargo profit. Thus, nodes where lots of cargo originates or is delivered have a high profit and are desirable for the shortest path algorithm to visit. Since cargo can only be delivered if both the origin and destination are on the path of a ship, this heuristic cannot guarantee that the path taken is actually one that has profitable cargo flows.

Empty Equipment handling We perform a similar process of adding profit to visits with empty equipment surpluses/deficits. The primary difference is that we do not know how much equipment can be loaded on to the vessel. Thus, we introduce a parameter $\ell^{Eqp} \in \mathbb{Z}^+$ that represents the amount of equipment to load or unload at a visit. We cannot guarantee the ship will have sufficient capacity to actually load or unload that amount of cargo, but this allows the shortest path algorithm to utilize visits with empty equipment, which might otherwise be ignored.

Shortest Path Implementation Since we take into account cargo and equipment profits in this heuristic, arcs can have costs or revenues associated with them, meaning the sign of all the arcs is not the same. We therefore use the Bellman-Ford algorithm to sequentially compute the shortest path for each vessel. After a shortest path is computed, we update a list of banned visits that may not be used again. Any visit that is banned is considered to have a distance of infinity to and from all visits, ensuring the shortest path algorithm does not choose it. It is possible that negative cycles are generated by the algorithm. We handle these by clearing the list of banned visits and starting the algorithm from the first vessel in the vessel ordering using updated cargo and equipment profit parameters. After each failure, we subtract 0.1 from the cargo profit parameter ℓ^{Cargo} and 50 from ℓ^{Eqp}, until these parameters reach zero. When both parameters are zero, the graph is guaranteed to not have any negative cycles since arcs reflect only sailing costs. In practice, this is only necessary on several instances.

4.1.3. Greedy heuristic (GH) The greedy heuristic (GH) chooses the most profitable outgoing arc from each visit based on the same profit calculations and parameters as SPH. Similar to SPH, GH does not allow vessels to visit the same visit more than once, and does this by storing a list of banned nodes after computing a greedy path for a vessel. The order the paths are generated in is the same as in SPH, as nearly every other ordering we tried resulted in failure of the algorithm to compute a solution. As in SPH, if a solution cannot be generated we fall back on DRH, although this did not happen in our experiments. We also tried creating a greedy heuristic that is not concerned with feasibility, however, we found that the solutions it generates tend to be of poor quality, as many vessels sail to the same phase-in visit and the SA must then completely construct new routes for those vessels.

4.2. Neighborhoods

We now describe the neighborhood operators for our SA algorithm. At each iteration of our SA, a neighborhood operator is chosen uniformly at random to modify the current solution. The first three neighborhoods we introduce have sizes that grow polynomially in the size of the problem, but remain too large to enumerate within a local search, whereas our last two involve generating paths through the graph, and can therefore generate exponentially many neighbors for a given solution.

We therefore use a sampling approach and generate neighbors at random rather than attempt an exhaustive search of the neighborhoods.

Visit addition A ship, s, is selected uniformly at random along with an arc (u, v) on the path of s. A new visit, w, is chosen such that an arc exists from u to w and from w to v, i.e. $w \in out(u)$ and $w \in in(v)$, and w is not already on the path of s. The visit w is then inserted into the path of s between u and v. If no such visit w exists, then the neighborhood performs no changes.

Visit removal A ship, s, is selected uniformly at random along with a visit on its path, u, such that u is neither the first or last visit on the path. Visit u is removed from the path if there exists an arc from the visit before u to the visit after u. If no such arc exists, the solution is not changed.

Visit swap Two ships s and s', $s \neq s'$ are chosen uniformly at random, and a visit u is selected from the path of s. If a visit w on the path of s' exists such that swapping u and w is possible, i.e. $In(u) \cap In(w) \neq \emptyset$ and $Out(u) \cap Out(w) \neq \emptyset$, and swapping u and v would not introduce a duplicated node on either path, then u and w are swapped between paths.

Random path completion (RPC) A random ship, s, is selected uniformly at random along with a visit, u, on its path. All visits subsequent to u are removed from the path of s, and are replaced with a random path from u to the graph sink. Each visit added to the random path must not already be on the path of s, to ensure there are no loops over flexible visits. If it is impossible to finish a random path without containing a loop, the random path is abandoned and the solution is not changed.

Demand destination completion (DDC) A random ship, s, is selected uniformly at random along with a visit on its path, u, from which demand is loaded. A demand is chosen that could be loaded at u, but cannot be delivered because none of its delivery visits are on the path of s. The neighborhood attempts to connect the current path to one of the destinations of the cargo using a breadth first search. Then, another breadth first search is started from the destination back to any subsequent visit on the path, v. If no such path exists, or such paths can only be created by introducing a duplicated node into the vessel's path, then the solution is left unchanged. All nodes in between u and v are deleted from the path and replaced with the nodes from the breadth first searches.

4.3. Objective Evaluation

We split our objective evaluation into two cases. The first case consists of a mathematical model to compute the objective in situations where flexible nodes or several cargo opportunities require a linear program in order to compute the objective. We then describe a second case where a greedy objective evaluation function can be used in certain situations.

4.3.1. Single ship mathematical model After applying a neighborhood operator to a solution and generating a candidate solution, the search procedure must update its objective value for the current solution. We exploit the fact that the paths of vessels are disjoint and only update the objective for the paths that a neighborhood operator changes. Note that since we admit infeasible solutions as described above, it is possible that some cargo is carried multiple times. However, the extra revenue that vessels can gain is significantly less than the penalty for multiple vessels calling the same visit.

The objective in the LSFRP consists of three components: the fixed costs for inflexible arcs and port fees, the cost of sailing on flexible arcs, and the profit from delivering cargo and equipment. The fixed costs are easy to compute since their costs are given by the constants of the problem. Computing the cost of using flexible arcs requires solving a simple scheduling problem for the vessel. Since the amount of time it takes to load cargo and equipment is taken into account at flexible visits, the scheduling of flexible arcs and the handling of cargo and equipment must be solved together. We therefore formulate a linear program to compute the objective and load the most profitable cargo along the vessel's path, and solve it using CPLEX. We adopt the same notation as in our mathematical model in Section 3.

We compute the cost of each ship's path independently according to the following mathematical formulation. We define the path of ship $s \in S$ as $\bar{V}_s = (v^s_1, \ldots, v^s_{n(s)})$, where $n(s)$ is the number of visits on the path of ship s. Additionally, let $\bar{V}^f_s = \bar{V}_s \cap V^f$ be the flexible visits used on the path, and $\bar{A}_s = ((v^s_1, v^s_2), (v^s_2, v^s_3), \ldots (v^s_{n(s)-1}, v^s_{n(s)}))$ be the sequence of arcs used on the path of ship s, and $\bar{A}^f_s = \bar{A}_s \cap A^f$ be the flexible arcs used on the path.

Since the ship's path is fully defined by $\bar{V}_s$, we can pre-compute which demands and equipment flows can be carried by ship s. We merge empty equipment flows into the demand structure of the problem. We can do this because the equipment source and destination ports are fixed by the ship's path. Let M^{LS}_s be the set of demand triplets relevant to the path of ship $s \in S$ merged with empty equipment triplets. Formally,

$$M^{LS}_s = \{(o,d,q) \in M \mid o \in \bar{V}_s \wedge \exists k \in d \text{ s.t. } k \in \bar{V}_s\} \bigcup_{q \in Q} E_q,$$

where M is the set of demand triplets, Q is the set of equipment types, and

$$E_q = \{(o,d,dc) \mid o \in V^{q+} \cap \bar{V}_s \wedge d \in V^{q-} \cap \bar{V}_s\},$$

is the set of origin-destination pairs for equipment that is available on the vessel path, in which V^{q+} and V^{q-} are the sets of empty equipment surplus and demand visits, respectively. For any $(o,d,dc) \in E_q, \forall q \in Q$, we let $r^{(o,d,dc)} = r^{Eqp}_q$ and $a^{(o,d,dc)} = u^{dc}_s$. This sets the revenue of delivering a

demand representing equipment to the per TEU equipment delivery revenue, and the amount of empty equipment available to be the dry capacity of the ship, respectively.

In order to keep track of the portion of the objective penalizing infeasibility, we let $g_i^s = |\{s' \in S \setminus \{s\} \mid i \in \bar{V}_{s'}\}|$, which is the number of paths in which visit i is contained, other than the path of ship s. Furthermore, the parameter h_i is equal to 1 iff $i \in \bigcup_{\ell \in L} R_\ell^{PI}$, meaning visit i is a phase-in visit.

Parameters Our LP uses the following parameters.

$\bar{V}_s, \bar{V}_s^f$	Sequence of visits on the path of ship $s \in S$.
$\bar{A}_s, \bar{A}_s^f$	Sequence of arcs on the path of ship $s \in S$.
Q	Set of equipment types. $Q = \{dc, rf\}$.
M_s^{LS}	Set of demands and equipment that can be carried by ship $s \in S$, consisting of demand triplets (o, d, q).
M_{si}^{LS}	Set of demands and equipment that can be carried by ship $s \in S$ at visit $i \in \bar{V}_s$, where $M_{si}^{LS} = \{(o, d, q) \in M_s^{LS} \mid i = o \vee i \in d\}$.
$M_{si,rf}^{LS}$	Set of reefer demands and equipment that can be carried by ship $s \in S$ at visit $i \in \bar{V}_s$, where $M_{si,rf}^{LS} = \{(o, d, q) \in M_{si}^{LS} \mid q = rf\}$.
$c_{sij}^{Sail} \in \mathbb{R}^+$	Fixed cost of vessel s utilizing arc $(i, j) \in A'$.
$c_{sij}^{VarSail} \in \mathbb{R}^+$	Variable hourly cost of vessel $s \in S$ utilizing arc $(i, j) \in A'$.
$c_i^{Mv} \in \mathbb{R}^+$	Cost of a TEU move at visit $i \in V'$.
$c_{si}^{Port} \in \mathbb{R}$	Port fee associated with vessel s at visit $i \in V'$.
$r^{(o,d,q)} \in \mathbb{R}^+$	Amount of revenue gained per TEU delivered for the demand triplet (o, d, q).
$\Delta_{ijs}^{Min}, \Delta_{ijs}^{Max} \in \mathbb{R}^+$	Minimum and maximum duration for vessel s to sail on flexible arc (i, j), respectively.
$t_i^E, t_i^X \in \mathbb{R}$	Enter and exit time at inflexible visit $i \in V'$, respectively.
$t_i^P \in \mathbb{R}$	Pilot time at visit $i \in V'$.
$t_{si}^{Mv} \in \mathbb{R}$	Move time per TEU for vessel s at visit $i \in V'$.
$a^{(o,d,q)} \in \mathbb{R}^+$	Amount of demand available for the demand triplet.
$u_s^q \in \mathbb{R}^+$	Capacity of vessel s for cargo type $q \in Q$.
g_i^s	Number of paths in which visit i is included, not including the path of ship s.
h_i	Equal to 1 iff visit i is a phase-in visit.

Variables Our LP uses the following variables.

$z_{si}^E \in \mathbb{R}^+$	Entrance time at visit $i \in \bar{V}_s$
$z_{si}^X \in \mathbb{R}^+$	Exit time at visit $i \in \bar{V}_s$
$x_s^{(o,d,q)} \in [0, a^{(o,d,q)}]$	Amount of demand $(o, d, q) \in M_s^{LS}$ carried.

Objective and constraints The LP is defined as follows.

$$\max \sum_{(o,d,q) \in M_s^{LS}} (r^{(o,d,q)} - c_o^{Mv} - \max_{k \in d}\{c_k^{Mv}\}) x_s^{(o,d,q)} \tag{19}$$

$$- \sum_{(i,j) \in \bar{A}_s} c_{sij}^{Sail} - \sum_{(i,j) \in \bar{A}_s^f} c_{sij}^{VarSail} (z_{sj}^X - z_{si}^E) - \sum_{i \in \bar{V}_s} c_{si}^{Port} \tag{20}$$

$$-\sum_{i\in\bar{V}_s} g_i^s\left((1-h_i)p+h_i p^{PI}\right) \tag{21}$$

$$\text{s.t.} \quad \Delta_{ijs}^{Min} \le z_{sj}^X - z_{si}^E \le \Delta_{ijs}^{Max} \qquad \forall (i,j)\in \bar{A}_s^f \tag{22}$$

$$z_{si}^E \le z_{si}^X \qquad \forall i\in\bar{V}_s \tag{23}$$

$$z_{si}^E = t_i^E \qquad \forall i\in\bar{V}_s\setminus\bar{V}_s^f \tag{24}$$

$$z_{si}^X = t_i^X \qquad \forall i\in\bar{V}_s\setminus\bar{V}_s^f \tag{25}$$

$$z_{si}^X - z_{si}^E - t_i^P - \sum_{(i,d,q)\in M_s^{LS}} t_{si}^{Mv} x_s^{(i,d,q)} - \sum_{\{(o,d,q)\in M_s^{LS}\mid i\in d\}} t_{si}^{Mv} x_s^{(o,d,q)} \ge 0 \qquad \forall i\in\bar{V}_s^f \tag{26}$$

$$\sum_{(o,d,q)\in M_{si}^{LS}} x_s^{(o,d,q)} \le u_s^{dc} \qquad \forall i\in\bar{V}_s \tag{27}$$

$$\sum_{(o,d,rf)\in M_{si,rf}^{LS}} x_s^{(o,d,rf)} \le u_s^{rf} \qquad \forall i\in\bar{V}_s \tag{28}$$

The objective is to maximize the container carrying profit (including equipment) in (19) minus the sailing costs, both flexible and inflexible, minus the port fees in (20), and minus the penalization of infeasibility in (21). We include the max term in (19) because d is a set of destinations, however these destinations actually represent the same underlying port with the same costs. We apply the penalization factor p for each path that visit i is contained in other than that of ship s if the visit is not a phase-in visit. If the visit is a phase-in visit, we apply the penalty p^{PI}. Note that the fixed sailing cost and port fees components of objective (20) are simply constants that can be calculated before solving the LP. These constants, with the addition of port fees and the penalty, are included for completeness, but are not passed to the LP solver.

Constraints (22) require that the sailing time on flexible arcs is between the minimum and maximum sailing speed of the ship. Constraints (23) ensure that the entrance time of a ship at a visit is earlier than its exit time. Constraints (24) and (25) fix the times of inflexible visits on the path. Note that we replace these variables with constants before passing the model into CPLEX to improve the solution speed. The loading and unloading time of containers at flexible visits is taken into account in constraints (26). Constraints (27) and (28) prevent the vessel from loading too many dry and reefer containers or reefer containers, respectively. The bounds of the variables are as defined.

If the model is temporally infeasible, we ignore cargo and equipment and penalize the objective by the constant p^T. We choose this over making the timing constraints soft because the sailing costs are not correctly computed in such a model, and even offers some incentive for infeasibility.

4.3.2. Greedy objective computation In certain situations it is possible to avoid calling CPLEX and thereby speed up the computation of the objective. If a vessel's path includes no

flexible visits and at no visit could the amount of cargo (both dry and reefer) loaded on to the vessel exceed its capacity, then we can use a simple greedy algorithm to load cargo onto the vessel. The greedy algorithm works by loading all available cargo at all visits on the path, and then fills the remaining capacity of the vessel with equipment. This will always be the optimal loading of cargo and equipment as long as the profit earned per TEU from carrying equipment is less than the profit per TEU of any demand. In practice this is true, since customer's cargo is preferred over empty containers. We then combine the cargo profit computed by the greedy algorithm with the sailing costs and port fees, which do not require an LP to compute.

5. Computational Study

We evaluate the performance of our MIP and the SA algorithm across a dataset of real-world and real-world inspired instances. We show that our novel approach to solving the LSFRP with cargo flows is ready for use in a decision support system, as the solution time is fast enough for human interaction (on the order of a few minutes). In addition, our SA algorithm is able to find high quality solutions on our industrial collaborator's reference scenario that earn twice the profit of the reference solution.

5.1. Benchmark

We created a benchmark set of instances containing one real world repositioning scenario with eleven ships, and one partially real-world scenario based on the routes in Figure 3. The rest of our benchmark set consists of scenarios that never took place, but were crafted using real liner shipping data to examine how our algorithms scale. Since all of our data in the benchmark is confidential information from our industrial collaborator, we have duplicated the confidential instances and perturbed the costs, revenues, amounts of cargo in demands, and randomly deleted/added demands to create a publicly available benchmark. We combine publicly available liner shipping data, such as ship information and port fees, from the ENERPLAN benchmark in Brouer et al. (2012) with randomly perturbed data from our industrial collaborator. We perturb all values not already contained in the ENERPLAN benchmark by $\pm 20\%$, as in Brouer et al. (2012), including non-cost/revenue related values such as port productivities, ensuring that no private data is contained in the dataset[4]. We have kept the schedules of the ships in the repositioning scenarios the same, as this is public information.

Tables 2 and 3 gives information about the instances in both the confidential and public datasets, respectively, along with the runtime of our MIP model in CPLEX 12.4 (IBM (2012)). The MIP runtimes are computed on a dual 6-core AMD Opteron 2425 HE machine with a maximum of 10GB

[4] Our public dataset is available at `http://www.decisionoptimizationlab.dk/index.php/datasets/17-research/datasets/59-lsfrpcf`.

of RAM per process. We allow CPLEX to use only a single processor. The tables give the instance ID, the number of ships, $|S|$, the number of nodes in the graph, $|V|$, the number of inflexible arcs, $|A^i|$, the number of flexible arcs, $|A^f|$, the number of demands, $|M|$, the number of ports with equipment surpluses or demands, $|E| = |\cup_{q\in Q} V^{q^*}|$, the number of SoS opportunities, $|SoS|$, and finally the runtime of CPLEX in seconds with a one hour timeout. Instances used in the training set for our parameter tuning are marked with an asterisk. The main difference between the confidential and public instances is their cost structure, as well as the number of demands and amount of cargo in each demand.

We present results for both the confidential and public datasets to show that our public dataset is able to capture the difficult components of the LSFRP, and is thus a viable benchmark for further study of the LSFRP. Note that the instance IDs correspond between the confidential and private instances, with the only difference being the cost structure of the instances, since the schedule information used in the instances is already public.

The instances range in size from 3 to 11 ships with various SoS and equipment opportunities made available in each instance. Our instances have between 30 and 379 nodes, and up to 11979 arcs. The number of demands can be as high as 1748, although most instances have less than 400 demands. The large number of demands often results in our model using a significant amount of memory due to the large number of variables generated from the multi-commodity flow. On those problems that do fit into memory, but still timeout, we suspect the problem to be the interaction between determining the paths of vessels and the multi-commodity flow, which must be recomputed for each new path a vessel is assigned.

Most instances have several hundred demands, with the largest confidential instance having 1748 demands and the largest public instance 1423 demands. CPLEX is able to solve 33 out of the 44 instances on both the confidential and the public datasets. Many instances are solved rather quickly, with over 25 confidential instances solved in under two minutes of CPU time and 17 of those instances requiring under ten seconds of computation time. Of the public instances, 28 are solvable in two minutes and 17 in under 10 seconds. The MIP approach does not scale past 8 ships on either dataset, as the graph sizes start to become large, and the number of variables increases as well. Many of these instances are unsolvable purely due to running out of memory, which gives hope for branch and price approaches. We encounter memory issues on instances that have many arcs and cargo/equipment demands, since a variable must be created for each demand on each arc.

In addition to solving our MIP model to optimality, we attempted to solve to a 5% and 10% gap, however the results of Tables 2 and 3 remain unchanged; instances that cannot be solved to optimality still cannot be solved to a 10% gap, and instances that can be solved are not solved significantly faster as a result. Since we aim to find good solutions to the LSFRP within ten minutes

Table 2 Confidential dataset instance information and solution time to optimality with CPLEX 12.4.

ID	$\|S\|$	$\|V\|$	$\|A^i\|$	$\|A^f\|$	$\|\Theta\|$	$\|E\|$	$\|SOS\|$	MIP
repos1c	3	30	94	0	27	0	1	0.04
repos2c	3	30	94	0	27	0	2	0.04
repos3c	3	37	113	0	25	0	2	0.03
repos4c	3	40	143	0	21	0	3	0.03
repos5c	3	47	208	0	24	0	3	0.05
repos6c	3	47	208	0	24	0	3	0.06
repos7c	3	53	146	0	51	0	4	0.06
repos8c	3	104	1015	121	67	6	3	1.92
repos9c	3	104	1015	121	67	9	3	1.91
repos10c	4	58	389	0	150	0	0	16.08
repos11c	4	62	389	40	150	6	0	14.50
repos12c	4	74	469	0	174	0	2	72.91
repos13c	4	80	492	0	186	0	4	231.47
repos14c	4	80	492	0	186	24	4	182.06
repos15c*	5	68	237	0	214	0	0	0.39
repos16c	5	103	296	0	396	0	5	0.95
repos17c*	6	100	955	0	85	0	0	5.41
repos18c*	6	133	1138	0	101	0	9	6.72
repos19c*	6	133	1138	0	101	33	9	5.68
repos20c	6	140	1558	0	97	0	4	313.55
repos21c*	6	140	1558	0	97	13	4	47.78
repos22c	6	140	1558	0	97	37	4	39.51
repos23c*	6	152	1597	162	103	71	9	19.79
repos24c	7	75	395	0	196	0	3	2.30
repos25c	7	77	406	0	195	0	0	2.69
repos26c	7	77	406	0	195	16	0	1.96
repos27c	7	78	502	0	237	0	0	94.48
repos28c	7	89	537	0	241	0	4	174.55
repos29c	7	89	537	0	241	19	4	186.38
repos30c	8	126	1154	0	165	0	0	2075.82
repos31c	8	126	1300	0	312	0	0	99.02
repos32c*	8	140	1262	0	188	0	3	487.93
repos33c	8	209	2211	453	213	50	3	548.11
repos34c	9	304	9863	0	435	0	0	Time
repos35c	9	357	11289	38	1075	118	4	Mem
repos36c*	9	364	11078	0	1280	0	4	Mem
repos37c*	9	371	10416	0	1270	114	7	Mem
repos38c*	9	373	11979	38	1280	126	4	Mem
repos39c*	9	379	10660	0	1371	0	7	Mem
repos40c	9	379	10660	0	1371	118	7	Mem
repos41c*	10	249	7654	0	473	0	0	Time
repos42c*	11	275	5562	0	1748	0	5	Time
repos43c*	11	320	11391	0	1285	0	0	Mem
repos44c	11	328	11853	0	1403	0	4	Mem

in order to create a decision support system with our algorithms that can be used in industry, heuristic methods are required.

5.2. SA Implementation

We implemented the SA algorithm described in Section 4 in C++11 (ISO/IEC (2011)). The implementation relies on CPLEX 12.4 for solving components of the objective function, as well as Google OR Tools (Google (2012)) for computing the assignment problem in the DRH heuristic. Our implementation is able to process over 700,000 SA iterations per second on smaller instances, where an

Table 3 Public dataset instance information and solution time to optimality with CPLEX 12.4.

ID	$\|S\|$	$\|V\|$	$\|A^i\|$	$\|A^f\|$	$\|\Theta\|$	$\|E\|$	$\|SOS\|$	MIP
repos1p	3	36	150	0	28	0	1	0.06
repos2p	3	36	150	0	28	0	2	0.06
repos3p	3	38	151	0	24	0	2	0.04
repos4p	3	42	185	0	20	0	3	0.04
repos5p	3	51	270	0	22	0	3	0.07
repos6p	3	51	270	0	22	0	3	0.08
repos7p	3	54	196	0	46	0	4	0.08
repos8p	3	108	1185	126	50	6	3	1.89
repos9p	3	108	1185	126	50	10	3	1.82
repos10p	4	58	499	0	125	0	0	74.85
repos11p	4	62	499	38	125	6	0	38.17
repos12p	4	74	603	0	145	0	2	106.63
repos13p	4	80	632	0	155	0	4	99.81
repos14p	4	80	632	0	155	24	4	97.15
repos15p	5	71	355	0	173	0	0	0.47
repos16p	5	106	420	0	320	0	5	1.08
repos17p	6	102	1198	0	75	0	0	4.64
repos18p	6	135	1439	0	87	0	9	6.79
repos19p	6	135	1439	0	87	33	9	8.18
repos20p	6	142	1865	0	80	0	4	13.84
repos21p	6	142	1865	0	80	13	4	23.04
repos22p	6	142	1865	0	80	37	4	17.67
repos23p	6	153	1598	159	89	71	9	19.58
repos24p	7	75	482	0	154	0	3	2.23
repos25p	7	77	496	0	156	0	0	3.19
repos26p	7	77	496	0	156	16	0	2.05
repos27p	7	79	571	0	188	0	0	1394.44
repos28p	7	90	618	0	189	0	4	1099.87
repos29p	7	90	618	0	189	19	4	1183.01
repos30p	8	126	1450	0	265	0	0	307.12
repos31p	8	130	1362	0	152	0	0	57.40
repos32p	8	144	1501	0	170	0	3	65.51
repos33p	8	212	2227	433	179	50	3	139.99
repos34p	9	304	10577	0	344	0	0	Time
repos35p	9	357	11284	35	874	118	4	Mem
repos36p	9	364	11972	0	1048	0	4	Mem
repos37p	9	371	11371	0	1023	114	7	Mem
repos38p	9	373	11972	35	1048	126	4	Mem
repos39p	9	379	11666	0	1109	0	7	Mem
repos40p	9	379	11666	0	1109	118	7	Mem
repos41p	10	249	8051	0	375	0	0	Time
repos42p	11	279	6596	0	1423	0	5	Time
repos43p	11	320	13058	0	1013	0	0	Mem
repos44p	11	328	13705	0	1108	0	4	Mem

iteration is an application of a neighborhood operator to the current solution and an update of the objective function, and 7,100 iterations per second on our largest instance, repos44c.

5.2.1. Parameter Tuning Our SA algorithm has many different parameters which can affect its performance, and in order to ensure a fair comparison of our SA against the MIP model, we perform parameter tuning on the SA. There are many suggestions in the literature for parameter settings of the components of SA algorithms, e.g., Johnson et al. (1989), Hoos and Stützle (2004). We have used these guidelines in our parameter tuning procedure, but are ultimately relying on the performance of our SA on a training set of instances to determine which parameters are the best

for the LSFRP. In order to avoid overtuning our algorithm to the instances of the LSFRP that we present, we tune our SA algorithm on a training set of 15 instances from the confidential dataset and validate the performance of the parameters on the entire set of instances. The instances were chosen at random from instances in which the SA algorithm using the GH heuristic was unable to find the optimal solution. The training set consists of 15 instances, which is a little over one third of our dataset. This is a standard amount in the machine learning and parameter tuning literature (Ansotegui et al. (2009)). The 15 instances used in our training set are marked with an asterisk in Table 2.

There are 13 parameters to tune in total. The feasible shortest paths heuristic has three parameters, γ, ℓ^{Cargo} and ℓ^{Eqp}, which describe the cost factor to use when estimating flexible arc costs, the amount of cargo profit to earn at the origin and destination visits of a demand, and the amount of equipment to "load" in the heuristic, respectively. The SA algorithm has seven parameters, α, t^{Init}, t^{Min}, r^{Itrs}, r^{Reheat}, β, $r^{Restart}$, which are the geometric temperature decrease factor, the starting SA temperature, the SA convergence temperature, the maximum number of non-improving iterations before convergence, the number of non-improving reheats before convergence, the reheating factor, and the number of reheats before resetting the current solution to the starting solution, respectively. Finally, there are three parameters that control the penalization of infeasible solutions in the objective. The parameters p, p^{PI}, and p^{T} are the penalization of multiple vessels utilizing the same visit in their paths, the penalty for two vessels using the same phase-in visit, and the penalty for a vessel's path being temporally infeasible. Note that p^{PI} is describing a case of p which we penalize separately because it tends to be a more difficult infeasibility for the SA to fix, and thus needs a higher penalization to ensure its repair.

We tune the SA algorithm by running each setting of each parameter to each value in its domain, independent of other parameters, ten times on each training instance, each time with a different random seed. Running each parameter configuration multiple times is necessary due to the stochastic nature of the SA algorithm. In order to avoid a combinatorial explosion of parameter settings, we assume that parameters are independent of each other. While this is clearly not a true assumption, it is the only way to perform parameter tuning across so many parameters without spending extraordinary amounts of CPU time. We then choose the best parameter value for each parameter based on the total profit earned by using each parameter.

Table 4 gives the discretized parameter domains we used during hand tuning, as well as the best value for each parameter for each initial heuristic with all SA neighborhoods enabled. We determined which parameter was the best by computing the total profit earned by each parameter across the training set.

Category	Parameter	Discretized domain values.	DRH	SPH	GH
GH/SPH	γ	0.0, 0.1, 0.25, 0.5, 0.75	0.25	0.25	0.25
	ℓ^{Cargo}	0.0, 0.1, 0.25, 0.5, 0.75, 0.95	0.95	0.95	0.95
	ℓ^{Eqp}	0, 10, 50, 100, 250, 500, 1000	100	100	100
SA	α	0.997, 0.998, 0.999, 0.9999, 0.99999, 0.999999	0.999999	0.999999	0.999999
	t^{Init}	1×10^4, 5×10^4, 1×10^5, 5×10^5, 1×10^6	1×10^6	1×10^6	5×10^5
	t^{Min}	1×10^{-8}, 1×10^{-10}, 1×10^{-12}, 1×10^{-15}	1×10^{-8}	1×10^{-8}	1×10^{-8}
	r^{Itrs}	500, 1000, 2500, 5000, 7500, 1×10^4	7500	10000	7500
	r^{Reheat}	1, 5, 10, 20	20	20	20
	β	0.1, 0.25, 0.5, 0.75	0.75	0.75	0.75
	$r^{Restart}$	1, 5, 10, 20	10	10	20
Penalization	p	$\{1,2,5,7\}\times10^5$, $\{1,2,5,7\}\times10^6$, 1×10^7	7×10^6	7×10^6	1×10^6
	p^{PI}	$\{1,2,5,7\}\times10^5$, $\{1,2,5,7\}\times10^6$, 1×10^7	7×10^6	5×10^6	5×10^6
	p^{T}	$\{1,2,5,7\}\times10^5$, $\{1,2,5,7\}\times10^6$, 1×10^7	1×10^5	1×10^5	7×10^5

Table 4 The discretized parameter domains used in hand tuning are given with parameters classified into several categories. The best parameter for each initial solution heuristic as determined through parameter tuning are given on the right side of the table.

	DRH	SPH	GH
# Best Obj.	29	30	13
# Worst Obj.	9	13	31
Obj. Average	-4.14×10^5	-1.34×10^6	-8.13×10^6
Obj. Median	-1.93×10^6	-1.93×10^6	-3.76×10^6

Table 5 Starting solution statistics for all three heuristics on the confidential dataset.

5.3. Initial Solution Heuristics Comparison

We compare the performance of the initial solution heuristics introduced in Section 4.1 across our dataset using the tuned parameters from Section 5.2.1. Table 5 describes the performance of the starting heuristics across the dataset.

The DRH and SPH heuristics achieve the best initial objectives out of the three heuristics on 29 and 30 of the instances, respectively (on many instances both heuristics return the same solution), while the GH heuristic only provides the best value on 13 instances. In order to determine the significance of the difference in the means of the solutions, we use a one-way ANOVA test (see, e.g., Tabachnick and Fidell (2012)) with the null hypothesis that the means of SPH, DRH and GH on the private dataset are not different from each other. We are unable to reject the hypothesis given the value $p = 0.581$. Although we can see that the solutions provided by each heuristic are not exactly the same, we do not have significant enough evidence to say that any one initial solution heuristic is better than another. Once the SA algorithm finishes, the answers tend to be relatively similar between the various starting heuristics.

5.4. Neighborhood Analysis

We perform an analysis of two of the neighborhoods from Section 4.2, the RPC neighborhood and the DDC neighborhood, in order to determine if they are beneficial to the SA algorithm. We do not

analyze the visit addition, removal and swapping neighborhoods because they are basic building blocks that any local search would need to be successful.

5.4.1. Random Path Completion We tune the parameters of the SA algorithm with and without the RPC neighborhood in order to determine whether the random paths generated are beneficial to the SA. We perform this experiment both with and without the DDC neighborhood, and solve each instance using 25 different seeds. Figures 6 and 7 show the performance of the SA algorithm using an initial solution generated by GH with the RPC neighborhood vs. not using the RPC neighborhood, both using the DDC neighborhood and without using DDC, respectively. We only show data using the GH initial heuristic since the performance of all three heuristics is similar after optimization. Points below the line $y = x$ in the scatter plots indicate better performance for the RPC neighborhood. The RPC neighborhood's usefulness is clear both with and without the DDC neighborhood. Not using the RPC neighborhood outperforms using the neighborhood only on several instances, and in many cases, the RPC neighborhood is able to find solutions that are orders of magnitude better than without the neighborhood. The line like structures in Figure 7 are due to multiple runs of instances that result a number of solutions with similar objectives.

With the DDC neighborhood, the average objective of GH with the RPC neighborhood is 8.6×10^6, versus an average objective of 7.3×10^6 without the RPC neighborhood. A t-test confirms the statistical significance of our findings, with $p < 1 \times 10^{-4}$, allowing us to reject the null hypothesis that the RPC neighborhood does not improve the solution quality. The difference in objective quality becomes even more pronounced when the DDC neighborhood is turned off. In this case, using the RPC neighborhood has an average objective of 7.4×10^6, but turning off RPC results in only -1.9×10^6.

We conclude that the RPC neighborhood is an important mechanism for the SA to explore new paths in the graph and avoid getting stuck in a local optimum. In contrast to the visit addition, removal and swap operators, which can help refine a solution, the RPC neighborhood creates large changes in solutions that are critical to good performance from our SA algorithm.

5.4.2. Demand Destination Completion We also test the effectiveness of the DDC neighborhood in order to see how much the neighborhood benefits the solution. Figure 8 plots the performance of the SA algorithm using initial solutions generated by GH with the DDC neighborhood vs. without the DDC neighborhood on each instance in the confidential dataset with 25 different seeds per instance. We hand tuned parameters for the SA for both with and without the neighborhood for fairness of comparison. Points below the line $y = x$ indicate a higher profit for the DDC neighborhood, whereas points above the line indicate that turning the neighborhood off provides a higher profit. The benefit of the DDC neighborhood is clearly demonstrated by the plot,

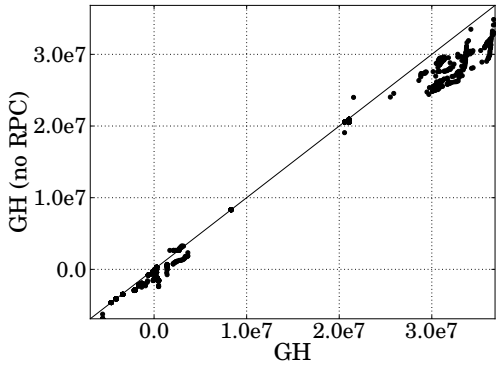

Figure 6 **Effectiveness of the RPC neighborhood with the DDC neighborhood**

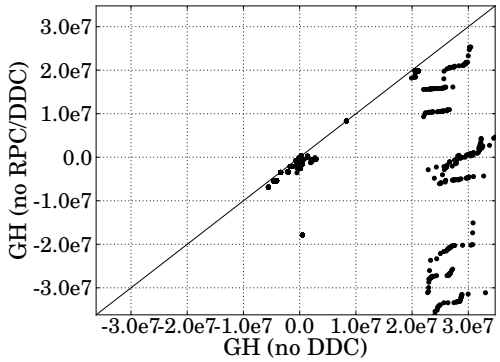

Figure 7 **Effectiveness of the RPC neighborhood without the DDC neighborhood**

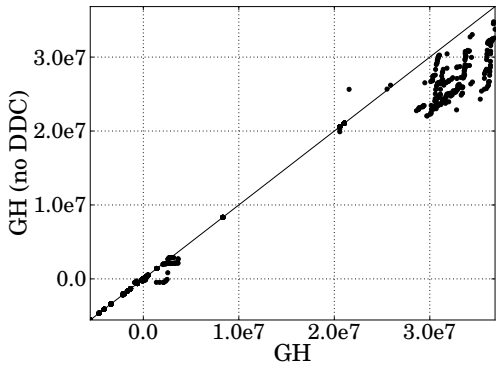

Figure 8 **Performance of the SA algorithm with and without the DDC neighborhood using the GH initial solution heuristic.**

with the majority of the instances lying below the line. Indeed, a t-test confirms the statistical significance of the result, allowing us to reject the null hypothesis that the mean of the SA performance with the DDC neighborhood is the same as without the neighborhood with a significance level of $p < 1 \times 10^{-4}$. The average objective across all instances and seeds with the neighborhood is $8.6{\times}10^6$, and without the neighborhood is $7.4{\times}10^6$, an improvement of 14%.

5.5. SA vs. MIP

We compare the objective value of the SA algorithm to the optimal value found by the MIP to determine whether our SA is finding good solutions.

Table 6 shows the objectives of the best solutions found by the parameter tuned SA algorithm out of 25 runs with the three starting heuristics versus the optimal solution found by the MIP on the confidential dataset. We also provide the optimality gap when the optimal solution is known. For those instances where no optimal solution is known (repos34c – repos44c), we only provide the objective of the solution found using SA. We allowed the SA algorithm to run for 10 minutes. The SA algorithm is able to find optimal solutions for 29 out of 33 instances where the optimal solution is known for both SPH and GH, and 28 instances for DRH. Even on the several instances where the optimal solution is not found (repos30c – repos33c), the average gap across these instances is only 0.0353. For a number of large instances we do not know the optimal solutions, and therefore cannot provide an optimality gap on these instances. We aim to remedy this with a column generation approach in future work. On these large instances (repos34c – repos44c), the GH initial solution heuristic tends to outperform the other heuristics, although this is not statistically significant.

Overall, the results between different initial solution heuristics are very similar. This indicates that our SA algorithm is not strongly affected by its starting point, a desirable quality in any local search approach. We note that although this table shows only the best solution found over 25 runs, the average case is also competitive, with SA still finding the optimal solution on 24 out of the 33 instances where the optimal solution is known for all three starting heuristics.

Table 7 shows the same data as Table 6 for the public dataset. A similar picture emerges, in which our SA algorithm is able to find optimal solutions across most of the dataset, with the initial solution heuristic being relatively irrelevant. Note that the objective values tend to be a bit lower than in the confidential data set. This is due to a combination of different ship fuel consumption profiles and a different demand structure than in the confidential dataset. Despite these small differences, the performance of the SA is comparable to its performance on the confidential dataset, indicating that our publicly available data is a good representation of the actual LSFRP.

We now show details of the performance of the SA on individual instances to give a better idea of the general performance of the algorithm. Figures 9, 10, and 11 show the average performance and standard error of SA using GH (solid black squares) as it solves several confidential instances with the best and worst SA performance (dashed black) and MIP performance (blue triangles) on instances repos6c repos10c and repos33c, respectively. The optimal solution value is shown as a horizontal black line. On instances repos6c, the MIP is able to solve the problem to optimality in well under a second and is therefore not visible in the graph. However, on instance repos10c, the MIP does not find a feasible solution until well after the SA has found an optimal solution. On

Table 6 The best objectives (in tens of thousands) and optimality gaps found with SA versus the optimal objective using all three starting heuristics out of 25 runs on each instance of the confidential dataset.

ID	Optimal	DRH		SPH		GH	
		Obj.	Gap	Obj.	Gap	Obj.	Gap
repos1c	-33.91	-33.91	0.000	-33.91	0.000	-33.91	0.000
repos2c	-33.91	-33.91	0.000	-33.91	0.000	-33.91	0.000
repos3c	-55.58	-55.58	0.000	-55.58	0.000	-55.58	0.000
repos4c	-6.30	-6.30	0.000	-6.30	0.000	-6.30	0.000
repos5c	0.44	0.44	0.000	0.44	0.000	0.44	0.000
repos6c	0.44	0.44	0.000	0.44	0.000	0.44	0.000
repos7c	83.20	83.20	0.000	83.20	0.000	83.20	0.000
repos8c	0.44	0.44	0.000	0.44	0.000	0.44	0.000
repos9c	0.44	-1.21	1.783	0.44	0.000	0.44	0.000
repos10c	205.76	205.76	0.000	205.76	0.000	205.76	0.000
repos11c	205.76	205.76	0.000	205.76	0.000	205.76	0.000
repos12c	210.34	210.34	0.000	210.34	0.000	210.34	0.000
repos13c	210.56	210.56	0.000	210.56	0.000	210.56	0.000
repos14c	210.56	210.56	0.000	210.56	0.000	210.56	0.000
repos15c	4.91	4.91	0.000	4.91	0.000	4.91	0.000
repos16c	4.91	4.91	0.000	4.91	0.000	4.91	0.000
repos17c	-16.64	-16.64	0.000	-16.64	0.000	-16.64	0.000
repos18c	-13.38	-13.38	0.000	-13.38	0.000	-13.38	0.000
repos19c	-13.38	-13.38	0.000	-13.38	0.000	-13.38	0.000
repos20c	-20.15	-20.15	0.000	-20.15	0.000	-20.15	0.000
repos21c	-20.15	-20.15	0.000	-20.15	0.000	-20.15	0.000
repos22c	-20.15	-20.15	0.000	-20.15	0.000	-20.15	0.000
repos23c	14.07	14.07	0.000	14.07	0.000	14.07	0.000
repos24c	-46.30	-46.30	0.000	-46.30	0.000	-46.30	0.000
repos25c	-41.07	-41.07	0.000	-41.07	0.000	-41.07	0.000
repos26c	-41.07	-41.07	0.000	-41.07	0.000	-41.07	0.000
repos27c	2.89	2.89	0.000	2.89	0.000	2.89	0.000
repos28c	2.67	2.67	0.000	2.67	0.000	2.67	0.000
repos29c	2.67	2.67	0.000	2.67	0.000	2.67	0.000
repos30c	0.62	0.58	0.063	0.58	0.063	0.58	0.063
repos31c	3.55	3.45	0.027	3.45	0.027	3.45	0.027
repos32c	1.57	1.52	0.031	1.52	0.031	1.52	0.031
repos33c	2.38	2.33	0.020	2.33	0.020	2.33	0.020
repos34c	-	36.48	-	37.01	-	36.48	-
repos35c	-	301.33	-	295.71	-	324.01	-
repos36c	-	322.79	-	313.38	-	341.49	-
repos37c	-	337.40	-	334.76	-	342.23	-
repos38c	-	316.00	-	310.27	-	344.45	-
repos39c	-	351.14	-	352.40	-	366.81	-
repos40c	-	357.07	-	358.92	-	368.32	-
repos41c	-	32.68	-	32.43	-	31.84	-
repos42c	-	307.09	-	308.85	-	318.07	-
repos43c	-	320.10	-	304.87	-	332.45	-
repos44c	-	315.14	-	306.06	-	344.63	-
Avg.	24.43	86.40	0.058	85.47	0.004	89.93	0.004
Std. dev.	81.26	140.38	0.305	138.79	0.013	146.33	0.013

instance repos33c, the MIP roughly tracks the performance of the worst SA instantiation, however the SA converges before finding the optimal solution, whereas the MIP continues to improve, eventually finding the optimal value at 548 seconds.

We also attempted to warm start the MIP using the best solution found by SA after running for 10 seconds using the GH initial solution heuristic. Although this approach is able to reduce the amount of time required to compute the optimal solution on several instances, such as on

Table 7 The best objectives (in tens of thousands) and optimality gaps found with SA versus the optimal objective using all three starting heuristics out of 25 runs on each instance of the confidential dataset.

ID	Optimal	DRH		SPH		GH	
		Obj.	Gap	Obj.	Gap	Obj.	Gap
repos1p	-39.83	-39.83	0.000	-39.83	0.000	-39.83	0.000
repos2p	-39.83	-39.83	0.000	-39.83	0.000	-39.83	0.000
repos3p	-61.77	-61.77	0.000	-61.77	0.000	-61.77	0.000
repos4p	-46.62	-46.62	0.000	-46.62	0.000	-46.62	0.000
repos5p	-8.21	-8.21	0.000	-8.21	0.000	-8.21	0.000
repos6p	-8.21	-8.21	0.000	-8.21	0.000	-8.21	0.000
repos7p	-11.49	-11.49	0.000	-11.49	0.000	-11.49	0.000
repos8p	-8.21	-11.54	0.405	-8.21	0.000	-11.54	0.405
repos9p	-8.21	-11.54	0.405	-8.21	0.000	-12.44	0.514
repos10p	137.61	137.61	0.000	137.61	0.000	137.61	0.000
repos11p	137.61	137.61	0.000	137.61	0.000	137.61	0.000
repos12p	138.55	138.55	0.000	138.55	0.000	138.55	0.000
repos13p	138.86	138.86	0.000	138.86	0.000	138.86	0.000
repos14p	138.86	138.86	0.000	138.86	0.000	138.86	0.000
repos15p	-36.59	-36.59	0.000	-36.59	0.000	-36.59	0.000
repos16p	-36.59	-36.59	0.000	-36.59	0.000	-36.59	0.000
repos17p	-9.36	-9.36	0.000	-9.36	0.000	-9.36	0.000
repos18p	5.22	5.22	0.000	5.22	0.000	5.22	0.000
repos19p	5.22	5.22	0.000	5.22	0.000	5.22	0.000
repos20p	-11.85	-11.85	0.000	-11.85	0.000	-11.85	0.000
repos21p	-11.85	-11.85	0.000	-11.85	0.000	-11.85	0.000
repos22p	-11.85	-11.85	0.000	-11.85	0.000	-11.85	0.000
repos23p	5.22	5.22	0.000	5.22	0.000	5.22	0.000
repos24p	-53.89	-53.89	0.000	-53.89	0.000	-53.89	0.000
repos25p	-53.13	-53.13	0.000	-53.13	0.000	-53.13	0.000
repos26p	-53.13	-53.13	0.000	-53.13	0.000	-53.13	0.000
repos27p	-28.20	-28.20	0.000	-28.20	0.000	-28.20	0.000
repos28p	-32.13	-32.13	0.000	-32.13	0.000	-32.13	0.000
repos29p	-32.13	-32.13	0.000	-32.13	0.000	-32.13	0.000
repos30p	5.72	4.93	0.138	5.06	0.115	5.35	0.064
repos31p	-12.08	-12.08	0.000	-12.08	0.000	-12.08	0.000
repos32p	-10.92	-10.92	0.000	-10.92	0.000	-10.92	0.000
repos33p	-10.92	-10.92	0.000	-10.92	0.000	-10.92	0.000
repos34p	-	-2.01	-	-2.01	-	-2.01	-
repos35p	-	132.01	-	124.98	-	135.82	-
repos36p	-	147.67	-	148.86	-	154.34	-
repos37p	-	129.29	-	128.08	-	133.84	-
repos38p	-	142.25	-	143.30	-	158.83	-
repos39p	-	146.22	-	143.63	-	149.44	-
repos40p	-	146.86	-	150.18	-	153.39	-
repos41p	-	-46.69	-	-51.33	-	-43.79	-
repos42p	-	242.78	-	243.28	-	244.28	-
repos43p	-	174.68	-	183.38	-	188.67	-
repos44p	-	175.46	-	176.63	-	186.70	-
Avg.	2.30	33.11	0.029	33.28	0.003	34.71	0.030
Std. dev.	60.33	84.44	0.098	84.76	0.020	86.70	0.111

repos13c where the time falls from 231.47 seconds to 134.02 seconds, and on repos29c where the optimal solution can be computed in 103.60 instead of 186.38 seconds, there are just as many instances where the solving time actually increases. On most instances, however, there is little change, indicating that the solutions provided are not strong enough to really help in pruning the search tree. Furthermore, this approach does not help CPLEX solve any instances that were not

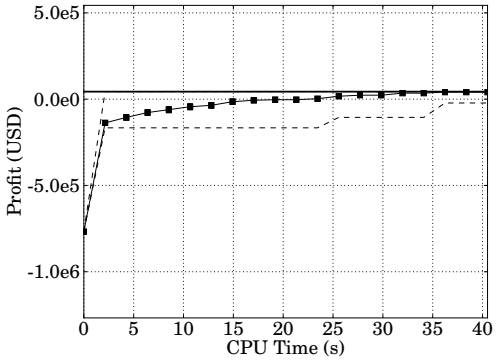

Figure 9 **Worst, average and best SA performance using GH on repos6c.**

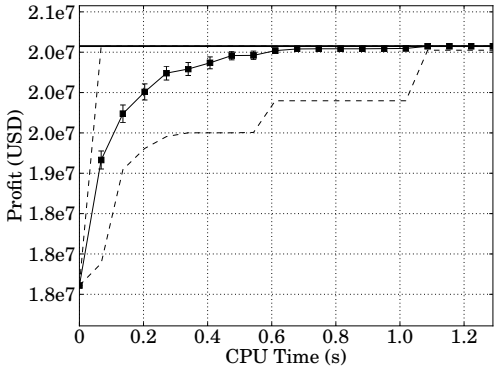

Figure 10 **Worst, average and best SA performance using GH on repos10c.**

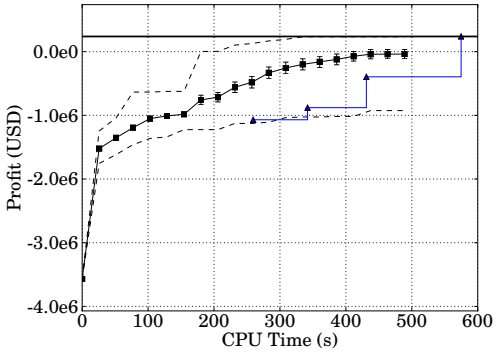

Figure 11 **Worst, average and best SA performance using GH on repos33c.**

previously solvable, i.e., no instances in which we experienced a timeout became solvable thanks to this approach.

5.6. Reference Scenario Comparison

We foresee the SA algorithm presented in this paper as being used in a decision support system for the LSFRP. In order to test whether or not the SA is effective at solving real problems, we compare the results of our algorithms with a reference scenario from our industrial collaborator. The scenario, instance repos42c, encompasses 11 vessels originating from 3 initial services. The

vessels seek to create a new service that visits the east coast of South America, Spain and the Middle East. The vessels have a single SoS each week that can be used from Tanjung Pelepas, Malaysia, to Jebel Ali, United Arab Emirates.

Since the reference solution to the scenario faced by our industrial partner was created in advance of the repositioning happening (as one would expect), the people who made it were at a disadvantage compared to our algorithm, which has a more complete view of the opportunities available during the full repositioning period. In order to counter act this unfairness, we calculate the profit of the reference solution under varying relaxations of restrictions present in our model.

Figure 12 shows the total profit earned by the reference solution as the size of the *demand delivery window* is increased. This window determines what visits may be used to deliver cargo. Our real-world data only specifies the date when demands were delivered, not the deadline for delivery. Thus, in our model, we use a value of ± 3 days for the demand delivery window, which means that any visit within three days of the delivery date is used as a feasible delivery visit for a demand. By relaxing this demand window to larger values, we allow the reference solution more flexibility as to where cargo gets delivered, which counter-acts the uncertainty planners had when creating the solution. The reference solution profit peaks at \$18,137,488 with a 14 day delivery window.

Figures 13 and 14 show the profit of the incumbent solution of the SA algorithm using the initial solution generated by GH in terms of the number of SA iterations and the CPU time, respectively. Error bars display the standard error across all 25 runs of the instance. The solid blue line shows the profit of the reference solution with a 14 day demand delivery window. SA-GH is able to find a solution with a better objective than the reference solution, even with a 14 day demand delivery window, in only 20 seconds or so of run time, and only 150 iterations. In the best case, SA finds a solution with an objective over \$13 million higher than the reference solution, and over \$11 million more in the average case. Even in the worst out of all 25 runs of the algorithm, we find a solution with a profit of over \$3 million more than the reference solution.

6. Conclusion

We presented a novel model of an important real-world problem, the LSFRP, and solved it using a MIP model on a constraint embedded graph and an SA approach. Our model takes into account all of the key aspects of the LSFRP, including liner shipping service construction constraints, cargo flows, empty equipment repositioning, cabotage restrictions, and sail-on-service opportunities, and maximizes the profit earned during repositioning. We evaluated our MIP and SA approaches, showing that not only does the SA scale to real-world sized problems, but it is also able to find a solution with a significantly higher profit than that of the reference solution from our industrial collaborator.

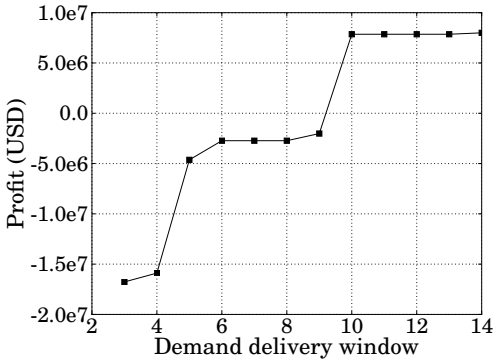

Figure 12 **The reference solution objective function.**

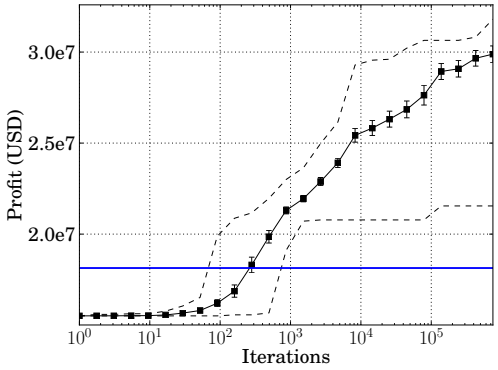

Figure 13 **SA with GH profit versus iterations on the reference instance.**

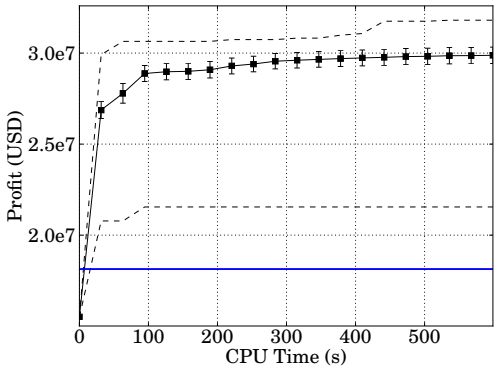

Figure 14 **SA with GH profit versus time on the reference instance.**

Our modeling techniques, especially our graph construction, could be applicable to other liner shipping problems, such as an extension to the vessel schedule recovery problem from Brouer et al. (2013) if SoS opportunities and flexible visits were included. Additionally, our SA approach, in particular our demand destination completion heuristic and initial solution heuristics, could be useful in other dual-layer flow problems, in which a multi-commodity flow is directed through the graph by the paths of vehicles.

For future work, we intend to use a branch and price framework to overcome scaling issues in our MIP model in order to solve large instances to optimality. Given the large amounts of money involved in the LSFRP, optimal solutions to repositioning problems can give liner carriers a critical edge over their competition.

Acknowledgments

We would like to thank our industrial collaborators Mikkel Muhldorff Sigurd and Shaun Long at Maersk Line for their support and detailed description of the fleet repositioning problem. Additionally, we would like to thank the anonymous reviewers of this work for their insightful comments that significantly improved this manuscript. This research is sponsored in part by the Danish Council for Strategic Research as part of the ENERPLAN research project.

Appendix A: Graph Formalization

In this appendix, we provide the details of our graph structure. The following parameters are used to define the graph.

S	Set of ships, indexed by s.
V'	Set of visits minus the graph sink.
V^i, V^f	Set of inflexible and flexible visits, respectively.
A^i, A^f	Set of inflexible and flexible arcs, respectively.
A'	Set of arcs minus those arcs connecting to the graph sink, i.e., $(i,j) \in A$, $i,j \in V'$.
L	Set of phase-in slots, where $\|L\| = \|S\|$, indexed by ℓ.
SoS	The set of SoS slots.
R_ℓ^{PI}	Set of visits of phase-in slot $\ell \in L$.
R_s^{PO}	Set of phase-out visits of vessel $s \in S$.
$O_o^{\{P,TS,T,E\}}$	Sets of parallel, transshipment, transit, and end visits, with $o \in SoS$.
V^R	Set of non-SoS inflexible visits, $V^R = \bigcup_{\ell \in l} R_\ell^{PI} \bigcup_{s \in s} R_s^{PO}$.
$\tau \in V$	Graph sink, which is not an actual visit.
TZ	Set of trade zones.
$z_i \in TZ$	Trade zone of visit $i \in V$.
$t_i^E \in \mathbb{R}$	Enter time at inflexible visit $i \in V'$.
$t_i^X \in \mathbb{R}$	Exit time at inflexible visit $i \in V'$.
d_{ij}^{min*}	Minimum time required for any ship to sail from visit i to j.
$A^{SD}(R)$	Set of arcs connecting subsequent visits in the visit set R.
A^{PO}	Set of arcs connecting phase-out slots to phase-in slots.
A^{PI}	Set of arcs from phase-in visits to same trade zone phase-out visits.
A^τ	Set of arcs from the final phase-in visit to the graph sink.
$\hat{A}_o^{In}$	Set of arcs connecting to the start nodes of $o \in SoS$.
$\hat{A}_o^{Out}$	Set of arcs extending from the end nodes of $o \in SoS$.
$\hat{A}_o^{PTS}$	Set of arcs connecting the parallel nodes to transshipment nodes of $o \in SoS$.
$\hat{A}_o^{TST}$	Set of arcs connecting transshipment nodes to transit nodes of $o \in SoS$.
$\hat{A}_o^{TT}$	Set of arcs between transit nodes of $o \in SoS$.
$\hat{A}_o^{EE}$	Set of arcs between sequential end nodes of $o \in SoS$.
$\hat{a}_o^{TE}$	Arc from the latest transit node in $o \in SoS$ to its earliest end node.

We define the set of inflexible nodes as $V^i = \bigcup_{\ell \in l} R_\ell^{PI} \bigcup_{s \in s} R_s^{PO} \bigcup_{o \in SoS} (O_o^P \cup O_o^T \cup O_o^{TS} \cup O_o^E)$. The set of flexible visits, V^f, contains all visits that have equipment surpluses/deficits such that $V^f \cap V^i = \emptyset$. In order

to formally define the set of arcs contained in the graph, let $follows(i,j) \in \mathbb{B}$ return *true* if and only if visit j is scheduled on any service to immediately follow visit i, with $i,j \in V^i$. In addition, we let $can\text{-}sail(i,j) \in \mathbb{B}$ be *true* if and only if $t_j^E \geq t_i^X + \Delta_{ij}^{min*}$, where $i,j \in V'$. This indicates whether or not it is possible to sail between two visits at the fastest speed of the fastest vessel in the model. Note that all of the arc sets are disjoint. We now formally define all of the previously mentioned sets of arcs.

$$
\begin{aligned}
A^{SD}(R) &= \{(i,j) \mid i,j \in R \wedge follows(i,j)\}, R \in \bigcup_{s\in S}\{R_s^{PO}\} \bigcup_{\ell \in L}\{R_\ell^{PI}\} \\
A^{PO} &= \{(i,j) \mid i \in \bigcup_{s\in S} R_s^{PO} \wedge j \in \bigcup_{\ell \in L} R_\ell^{PI} \wedge can\text{-}sail(i,j)\} \\
A^{PI} &= \{(i,j) \mid i \in \bigcup_{\ell \in L} R_\ell^{PI} \wedge j \in \bigcup_{s\in S} R_s^{PO} \wedge z_i = z_j \wedge can\text{-}sail(i,j)\} \\
A^{\tau} &= \{(i,\tau) \mid i \in \bigcup_{\ell \in L} \arg\max_{i' \in R_\ell^{PI}}\{t_{i'}^X\}\} \\
A^{f} &= \{(i,j) \mid \big((i \in V^f \vee j \in V^R) \wedge (i \in V^R \vee j \in V^f) \wedge (i \in V^f \vee j \in V^f)\big) \wedge z_i = z_j\} \\
\hat{A}_o^{In} &= \{(i,j) \mid i \in \bigcup_{s\in S} R_s^{PO} \wedge j \in (O_o^P \cup O_o^{TS}) \wedge can\text{-}sail(i,j)\} \\
&\quad \bigcup \{(i,j) \mid i \in V^f \wedge j \in (O_o^P \cup O_o^{TS}) \wedge z_i = z_j \wedge can\text{-}sail(i,j)\} \\
\hat{A}_o^{Out} &= \{(i,j) \mid i \in O_o^E \wedge j \in \left(\bigcup_{\ell \in L} R_\ell^{PI} \bigcup_{o' \in \{SoS \setminus o\}} (O_{o'}^P \cup O_{o'}^{TS})\right) \wedge can\text{-}sail(i,j)\} \\
&\quad \bigcup \{(i,j) \mid i \in O_o^E \wedge j \in V^f \wedge z_i = z_j \wedge can\text{-}sail(i,j)\} \\
\hat{A}_o^{PTS} &= \{(i,j) \mid i \in O_o^P \wedge j \in O_o^{TS} \wedge follows(i,j)\} \\
\hat{A}_o^{TST} &= \{(i,j) \mid i \in O_o^{TS} \wedge j \in O_o^{T} \wedge follows(i,j)\} \\
\hat{A}_o^{TT} &= \{(i,j) \mid i,j \in O_o^{T} \wedge follows(i,j)\} \\
\hat{A}_o^{EE} &= \{(i,j) \mid i,j \in O_o^{E} \wedge follows(i,j)\} \\
\hat{a}_o^{TE} &= (\arg\max_{i \in O_o^T}\{t_i^X\}, \arg\min_{j \in O_o^E}\{t_j^E\})
\end{aligned}
$$

The set of all arcs in the graph, A, is therefore defined by

$$
\begin{aligned}
A = &\bigcup_{s\in S}\left(A^{SD}(R_s^{PO})\right) \bigcup_{\ell \in L}\left(A^{SD}(R_\ell^{PI})\right) \cup A^{PI} \cup A^{f} \cup A^{\tau} \\
&\bigcup_{o \in SoS}\left(\hat{A}_o^{In} \cup \hat{A}_o^{Out} \cup A_o^{ST} \cup \hat{A}_o^{TT} \cup \hat{A}_o^{EE} \cup \hat{a}_o^{TE}\right).
\end{aligned}
$$

References

Agarwal, R., Ö. Ergun. 2008. Ship scheduling and network design for cargo routing in liner shipping. *Transportation Science* **42**(2) 175–196.

Álvarez, J.F. 2009. Joint routing and deployment of a fleet of container vessels. *Maritime Economics and Logistics* **11**(2) 186–208.

Andersen, M.W. 2010. Service Network Design and Management in Liner Container Shipping Applications. Ph.D. thesis, Technical University of Denmark, Department of Transport.

Ansotegui, C., M. Sellmann, K. Tierney. 2009. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. Ian P. Gent, ed., *Principles and Practice of Constraint Programming (CP 2009)*, *LNCS*, vol. 5732. Springer, 142–157.

Brouer, B.D., J.F. Alvarez, C.E.M. Plum, D. Pisinger, M.M. Sigurd. 2012. A base integer programming model and benchmark suite for liner shipping network design. *Transportation Science* .

Brouer, B.D., J. Dirksen, D. Pisinger, C.E.M Plum, B. Vaaben. 2013. The Vessel Schedule Recovery Problem (VSRP) – A MIP model for handling disruptions in liner shipping. *European Journal of Operational Research* **224**(2) 362–374.

Christiansen, M. 1999. Decomposition of a combined inventory and time constrained ship routing problem. *Transportation Science* **33**(1) 3–16.

Christiansen, M., K. Fagerholt, B. Nygreen, D. Ronen. 2007. Maritime transportation. *Handbooks in operations research and management science* **14** 189–284.

Christiansen, M., K. Fagerholt, B. Nygreen, D. Ronen. 2013. Ship routing and scheduling in the new millennium. *European Journal of Operational Research* **228**(3) 467–483.

Christiansen, M., K. Fagerholt, D. Ronen. 2004. Ship routing and scheduling: Status and perspectives. *Transportation Science* **38**(1) 1–18.

Clausen, J., A. Larsen, J. Larsen, N.J. Rezanova. 2010. Disruption management in the airline industry–concepts, models and methods. *Computers & Operations Research* **37**(5) 809–821.

Google. 2012. Google OR-Tools. http://code.google.com/p/or-tools/.

Hoos, H.H., T. Stützle. 2004. *Stochastic local search: Foundations & applications*. Morgan Kaufmann.

IBM. 2012. IBM CPLEX Reference manual and user manual. V12.4.

ISO/IEC. 2011. Information technology – Programming languages – C++, Third Edition. ISO/IEC 14882:2011, International Organization for Standardization / International Electrotechnical Commission, Geneva, Switzerland.

Johnson, D.S., C.R. Aragon, L.A. McGeoch, C. Schevon. 1989. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Operations Research* **37**(6) 865–892.

Jorgensen, R. 2011. Slow steaming – The full story. `http://www.maersk.com/Innovation/WorkingWithInnovation/Documents/Slow%20Steaming%20-%20the%20full%20story.pdf`. A.P. Moller-Maersk Group. Accessed: 27/3/2013.

Kelareva, E., K. Tierney, P. Kilby. 2013. CP Methods for Scheduling and Routing with Time-Dependent Task Costs. C. Gomes, M. Sellmann, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, *Lecture Notes in Computer Science*, vol. 7874. Springer Berlin Heidelberg, 111–127.

Kirkpatrick, S., CD Gelatt, MP Vecchi. 1983. Optimization by simulated annealing. *Science* **220** 671–680.

Kohl, N., A. Larsen, J. Larsen, A. Ross, S. Tiourine. 2007. Airline disruption management–perspectives, experiences and outlook. *Journal of Air Transport Management* **13**(3) 149–162.

Korsvik, J.E., K. Fagerholt, G. Laporte. 2011. A large neighbourhood search heuristic for ship routing and scheduling with split loads. *Computers & Operations Research* **38**(2) 474 – 483.

Kuhn, H.W. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly* **2**(1-2) 83–97.

Lourenço, H., O. Martin, T. Stützle. 2003. Iterated Local Search. *Handbook of Metaheuristics* 320–353.

Meyer, J., R. Stahlbock, S. Voß. 2012. Slow steaming in container shipping. *45th Hawaii International Conference on System Science (HICSS), 2012*. IEEE, 1306–1314.

Powell, B.J., A.N. Perakis. 1997. Fleet deployment optimization for liner shipping: An integer programming model. *Maritime Policy and Management* **24**(2) 183–192.

Ronen, D. 1983. Cargo ships routing and scheduling: Survey of models and problems. *European Journal of Operational Research* **12**(2) 119–126.

Stahlbock, R., S. Voß. 2008. Operations research at container terminals: a literature update. *OR Spectrum* **30**(1) 1–52.

Steenken, D., S. Voß, R. Stahlbock. 2004. Container terminal operation and operations research – a classification and literature review. *OR spectrum* **26**(1) 3–49.

Tabachnick, B.G., L.S. Fidell. 2012. *Using multivariate statistics*. Pearson.

Taheri, J., A.Y. Zomaya. 2007. A simulated annealing approach for mobile location management. *Computer communications* **30**(4) 714–730.

Tierney, K., A.J. Coles, A.I. Coles, C. Kroer, A.M Britt, R.M. Jensen. 2012. Automated planning for liner shipping fleet repositioning. L. McCluskey, B. Williams, J.R. Silva, B. Bonet, eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*. 279–287.

Tierney, K., R. M. Jensen. 2012. The Liner Shipping Fleet Repositioning Problem with Cargo Flows. Hao Hu, Xiaoning Shi, Robert Stahlbock, Stefan Voß, eds., *Computational Logistics*, *Lecture Notes in Computer Science 7555*, vol. 7555. Springer, 1–16.

United Nations Conference on Trade and Development (UNCTAD). 2012. Review of maritime transport.

Wang, S., Q. Meng. 2012. Sailing speed optimization for container ships in a liner shipping network. *Transportation Research Part E: Logistics and Transportation Review* **48**(3).