A multi-start local search heuristic for ship scheduling—a computational study

Geir Brønmo\textsuperscript{a,b,*}, Marielle Christiansen\textsuperscript{b}, Kjetil Fagerholt\textsuperscript{c}, Bjørn Nygreen\textsuperscript{b}

\textsuperscript{a}MARINTEK, Trondheim, Norway
\textsuperscript{b}Section of Managerial Economics and Operations Research, Norwegian University of Science and Technology, Trondheim, Norway
\textsuperscript{c}Department of Marine Technology, Norwegian University of Science and Technology, Trondheim, Norway

Available online 28 June 2005

Abstract

We present a multi-start local search heuristic for a typical ship scheduling problem. A large number of initial solutions are generated by an insertion heuristic with random elements. The best initial solutions are improved by a local search heuristic that is split into a quick and an extended version. The quick local search is used to improve a given number of the best initial solutions. The extended local search heuristic is then used to further improve some of the best solutions found. The multi-start local search heuristic is compared with an optimization-based solution approach with respect to computation time and solution quality. The computational study shows that the multi-start local search method consistently returns optimal or near-optimal solutions to real-life instances of the ship scheduling problem within a reasonable amount of computation time.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Maritime transportation; Integer programming; Set partitioning; Routing

1. Introduction

During the last couple of decades we have witnessed tough competition between shipping companies, where the profit margins have been squeezed to a minimum. This situation has been amplified by the

* Corresponding author.

E-mail addresses: geir.bronmo@iot.ntnu.no (G. Brønmo), marielle.christiansen@iot.ntnu.no (M. Christiansen), kjetil.fagerholt@ntnu.no (K. Fagerholt), bjorn.nygreen@iot.ntnu.no (B. Nygreen).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.05.017
consolidation in the manufacturing sector. The growth in the size of the main actors on the demand side for sea transport services has resulted in increased market power for the cargo owners or shippers. In order to reduce this imbalance, there have been many mergers among shipping companies in the last decade. Many of them have also entered into pooling and collaboration efforts in order to increase their market power and gain economics of scale. These trends of mergers and pooling collaborations result in larger operating fleets and more information spread over several geographically separated areas. This means that it becomes much harder to decide on a good fleet schedule using simple manual planning methods that have been widely used in the past. Further, the increasing competition in the shipping industry forces the companies to manage their fleets efficiently, simply to remain in business. These issues are some of the reasons for an increased need for advanced decision-support systems within ship scheduling, something that will probably continue to increase in the future. In Christiansen et al. [1], further arguments are given for the accelerating needs for and benefits from such systems.

In shipping, we usually distinguish between three general modes of operation: industrial, tramp and liner; see [2]. In industrial shipping, the cargo owner or shipper controls the ships. Industrial operators strive to minimize the costs of shipping their cargoes. Tramp ships follow the available cargoes, like a taxi. A tramp shipping company may have a certain amount of contract cargoes that it is committed to carry, and tries to maximize the profit from optional cargoes. Liners operate according to a published itinerary and schedule similar to a bus line. During the last few decades there has been a shift from industrial to tramp shipping. Perhaps the main reason for this is that many cargo owners are now focusing on their core business and have outsourced other activities, such as transportation, to independent shipping companies. The different modes of operation are described in more detail in [1], a paper which discusses other reasons for the shift from industrial to tramp shipping as well.

In this paper, we will consider short-term scheduling problems within tramp shipping. In the literature, we observe that most contributions are related to industrial shipping, while only a few have considered the tramp market; see surveys [1,3,4]. The main reason for the minimal attention to tramp scheduling in the literature may be that historically the tramp market was operated by a large number of small operators. However, this is not the case anymore. Appelgren [5,6] described a typical tramp ship scheduling problem already during the end of the 1960s. This work was the first to use a Dantzig–Wolfe decomposition approach for routing and scheduling, an approach that has been used later in numerous other studies.

The purpose of this paper is to present a new multi-start local search heuristic for a tramp ship scheduling problem. We compare the heuristic with a set partitioning approach regarding computational time and solution quality. The heuristic is designed for use in a decision-support system for tramp ship scheduling. Decision-support tools for the tramp market are in great demand, and this study will contribute by analyzing whether the proposed heuristic sufficiently meets the requirements of the tramp shipping industry.

The rest of the paper is organized as follows: In Section 2 we present the ship scheduling problem that is addressed. The set partitioning approach is briefly described in Section 3, while the multi-start local search heuristic is described in Section 4. A computational study based on real tramp ship scheduling problems from several shipping companies is presented in Section 5. This section includes case descriptions, computational results and a discussion of the results. Our conclusions follow in Section 6.
2. Problem description

The short-term ship scheduling problem that is studied corresponds to a maximum profit problem with pickups and deliveries of bulk cargoes in the tramp market. A tramp shipping company often engages in contracts of affreightment (COA). These are contracts to carry specified quantities of cargo between specified ports within a specific time frame for an agreed payment per tonne. In some cases the controlled fleet may have insufficient capacity to serve all COA-cargoes during the planning horizon. In such a case some of the cargoes can be serviced by spot charters, which are ships chartered for a single voyage. In addition to the COA-cargoes, a tramp shipping company may take optional spot cargoes. These cargoes will be picked up at a given loading port and delivered to a corresponding unloading port if the shipping company finds it profitable. Time windows are usually imposed for the loading of the cargoes, and sometimes also for the unloading.

For some operations, the ship capacities, the cargo type and quantities are such that the ships may carry multiple cargoes simultaneously. This means that a new loading port can still be visited with some cargoes onboard. We assume that the cargoes can be loaded onboard irrespective of the type of product already onboard, given that the ship’s capacity is not exceeded. In addition, not all ships can visit all ports and take all cargoes.

At the beginning of the planning period, a ship may be in a port or at a point at sea, while at the end of the planning period we assume that the ship is in the last planned unloading port. If the ship is not used during the planning period, the start and end positions of the ship is assumed to be the same.

In the short-term, it is of no interest to plan a change in the fleet size. Therefore, we are concerned with the operations of a given number of ships within the planning horizon. The fixed costs can be disregarded as they have no influence on the planning of optimal routes and schedules. We assume a heterogeneous fleet of ships with specific ship characteristics including different cost structures and load capacities. The ships are charged port and channel tolls when visiting ports and passing channels and these costs depend on the size of the ship. The remaining variable sailing costs consist mainly of fuel and diesel oil costs. We assume that the sailing costs do not depend on the load onboard the ship. Normally, tramp shipping companies seek to maximize the profit of their activity, so we use this objective when optimizing the ship schedules.

3. The set partitioning approach

The problem described in Section 2 can be formulated as an arc-flow model, see [7], and solved directly by use of standard commercial optimization software for mixed integer linear programming. Due to its complexity, only small-sized data instances can be solved to optimality. In order to establish real-life benchmarks for the heuristic, the model is formulated as a set partitioning problem instead, with columns corresponding to feasible ship schedules. All columns are pre-calculated and transferred to the set partitioning problem before it is solved. This is an approach that has been widely used for solving ship scheduling problems; see for instance [8–12].

Section 3.1 gives the set partitioning formulation of the short-term tramp ship planning problem, while the column generation algorithm is described in Section 3.2.
3.1. The tramp ship scheduling problem formulated as a set partitioning problem

The tramp ship scheduling problem can be formulated as a set partitioning problem with variables that correspond to feasible ship schedules. In the mathematical description of the problem each cargo is represented by an index \( i \). We need to partition the set of cargoes, \( \mathcal{N} \), into two subsets, \( \mathcal{N} = \mathcal{N}_C \cup \mathcal{N}_O \), where \( \mathcal{N}_C \) is the set of cargoes the shipping company has committed itself to carry, while \( \mathcal{N}_O \) represents the optional spot cargoes. Further, let \( \mathcal{V} \) be the set of ships in the fleet indexed by \( v \). Let us assume that for each ship \( v \), a set of candidate schedules is available, denoted \( \mathcal{R}_v \), and a specific schedule is indexed by \( r \). For each schedule \( r \), the visiting sequence of all the loading and unloading ports and the associated arrival times at each port are known. For a given combination of cargo set and ship, the schedule with maximum profit is represented by a variable in the model. \( P_{vr} \) is the profit from carrying the cargoes on schedule \( r \) by ship \( v \), while \( \pi_i \) is the profit if cargo \( i \) is serviced by a spot charter. This spot charter profit can be either positive or negative. Constant \( A_{ivr} \) is equal to one if schedule \( r \) for ship \( v \) services cargo \( i \) and zero otherwise.

In the mathematical formulation, we use the following types of variables. The binary variable \( y_{vr} \), \( v \in \mathcal{V} \), \( r \in \mathcal{R}_v \), equals one if ship \( v \) sails schedule \( r \) and zero otherwise, while the variable \( s_i \), \( i \in \mathcal{N}_C \), is equal to one if the COA-cargo \( i \) is serviced by a spot charter and zero otherwise.

The set partitioning formulation of the tramp ship scheduling problem with spot charters can then be given as follows:

\[
\begin{align*}
\text{max} & \quad \sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}_v} P_{vr} y_{vr} + \sum_{i \in \mathcal{N}_C} \pi_i s_i, \\
\sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}_v} A_{ivr} y_{vr} + s_i &= 1, \quad \forall i \in \mathcal{N}_C, \\
\sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}_v} A_{ivr} y_{vr} &\leq 1, \quad \forall i \in \mathcal{N}_O, \\
\sum_{r \in \mathcal{R}_v} y_{vr} &= 1, \quad \forall v \in \mathcal{V}, \\
y_{vr} &\in \{0, 1\}, \quad \forall v \in \mathcal{V}, \ r \in \mathcal{R}_v, \\
s_i &\in \{0, 1\}, \quad \forall i \in \mathcal{N}_C.
\end{align*}
\]

The objective function (1) maximizes the profit gained by operating the fleet in the tramp market (or actually the marginal contribution, since fixed costs are excluded from the formulation). The terms are divided into the profit gained by (a) operating the fleet and (b) servicing the cargoes by spot charters. Constraints (2) ensure that all cargoes that the shipping company has committed itself to carry are serviced, either by a ship in the company’s fleet or by a spot charter. The corresponding constraints for the optional cargoes are given in (3). Note that the equality sign in (2) is replaced by an inequality in (3) since these cargoes do not have to be carried. Due to the branch-and-bound algorithm, it might be useful to insert an explicit slack variable into constraints (3). The set packing constraints (3) will then become set partitioning constraints and the model becomes a pure set partitioning model. Constraints (4) assure that each ship in the fleet is assigned exactly one schedule (column). We have introduced schedules containing...
no real ports for each of the vessels. This means that if a ship is not used during the planning period, such an artificial schedule is chosen in the optimal solution. Constraints (5) and (6) impose the binary requirements on the variables. According to (2), the $s_i$ variables do not need to be defined as binary since the schedule variables are binary. However, normally the binary definition is preferable with regard to the branch-and-bound search.

In the short-term tramp shipping problem described, we assumed that the cargoes and ships are such that the ships may carry multiple cargoes simultaneously. For some fleets operating in particular segments, only full loads are possible and feasible. This means that each cargo is loaded in a loading port and the cargo is transported directly to its unloading port. The set partitioning formulations (1)–(6) hold for both multiple and full cargo problems.

### 3.2. Generation of the ship schedules

A column vector in the set partitioning formulation given in Section 3.1 corresponds to one of the feasible schedules belonging to the set $R_v$ for ship $v$. Each column in the set partitioning formulation contains information about the actual cargoes in the schedule and the ship that is used. The route and schedule are not given in the column, but are the basis for the profit coefficient in the objective function.

Here, we generate all columns a priori by use of an enumeration technique. First, all feasible schedules for each cargo set and ship combination are calculated. Each schedule contains the sequence of loading ports and unloading ports of each cargo in the cargo set. Second, the loading port is visited before its corresponding unloading port in the schedule, but not necessarily in succession. Finally, each cargo has to be serviced within its specified time windows at the loading and unloading ports and the ship capacity constraints have to be fulfilled. When all the feasible schedules are generated for a cargo set and ship combination, the one with maximum profit is the basis for the column transferred to the set partitioning model.

As mentioned, the set partitioning model also applies for a case where only full loads are possible. In the schedule generation phase, the number of feasible schedules for a specified number of cargoes is limited compared with the multiple cargo case. For instance, with three cargoes the maximum number of different schedules is six. Normally, several of these schedules are not feasible due to the time windows.

It is possible to generate all columns a priori for many of the real instances of the tramp ship scheduling problem, since they are often relatively small or tightly constrained [1]. Due to the time windows and capacity constraints, just a small part of the possible cargo combinations is feasible. However, the approach is time consuming compared with an efficient heuristic and unsuitable when solving very large problems. This approach will be used as a benchmark for the heuristic presented in Section 4, and computational results for many real instances will be presented in Section 5.

### 4. A multi-start local search heuristic

This section describes a multi-start local search heuristic for the tramp ship scheduling problem. In multi-start local search heuristics a number of different initial solutions are generated by a constructive heuristic and improved by local search. The framework is thoroughly described in [13]. Multi-start heuristics are efficient for problems where solutions can be easily constructed and where it is difficult to make local search neighborhoods that can move far in the feasible search space. Ship scheduling problems normally have these characteristics, since they are often tightly constrained.
To the authors’ knowledge there has been limited research on local search-based heuristics for ship scheduling problems. However, within related routing problems, such as the vehicle routing problem with time windows (VRPTW), the dial-a-ride problem with time windows and the multi-vehicle pickup and delivery problem with time windows, numerous heuristics have been developed. In later years tabu search heuristics have been considered by many authors to be the most efficient approach for a number of problems, see for instance [14–16].

Multi-start local search heuristics for routing and scheduling problems have not received much attention in the literature. However, Bräysy et al. [17] recently presented an efficient multi-start local search heuristic for the VRPTW, giving competitive results. Also, Section 5 shows that multi-start local search heuristics can also be efficient in solving ship scheduling problems.

Section 4.1 deals with the generation of initial solutions. We construct a large number of such solutions by a partly randomized insertion heuristic. Normally, in a multi-start local search every initial solution will be improved by a local search heuristic. In order to speed up the heuristic, we have decided to improve only a selection of the best initial solutions, and split the local search heuristic into a quick and an extended version. First, the selected initial solutions are improved by the quick local search heuristic, described in Section 4.2. Then a number of the best solutions from the quick local search are chosen, and these solutions are improved by the extended local search heuristic. Section 4.3 describes the extended local search heuristic, while an overview of the heuristic is given in Section 4.4.

### 4.1. Generating initial solutions

The set of initial solutions is very important for the performance of a multi-start local search heuristic. It should contain a combination of high-quality and diverse solutions bringing the search as close as possible to the optimum. Ronen [18] showed that a biased random algorithm provided good solutions to a simpler problem from industrial shipping. In this paper we generate diverse initial solutions by constructing a part of each solution randomly by using a similar biased random insertion procedure. The rest of each solution is then constructed using a deterministic insertion heuristic that imports high-quality to the solution.

A large number of initial solutions are generated by repeating the following procedure a specified number of times. First, a percentage of the cargoes, specified by the parameter $\text{RANDOM\_P}$, is subjected to the biased random insertion procedure. The procedure is biased since the sequence of port nodes is determined deterministically. The biased random procedure is as follows:

**Step I**: Make a list $L$ of the unassigned cargoes.

**Step II**: Until a percentage $\text{RANDOM\_P}$ of the cargoes is removed from $L$ do:

- **Step IIa**: Randomly select a cargo $i$ from $L$, and randomly select a ship $v$.
- **Step IIb**: Assign cargo $i$ to ship $v$ by finding the best possible insertion in the given schedule.
- **Step IIc**: If the assignment is feasible, remove cargo $i$ from $L$.
- **Step IId**: If the assignment is infeasible, select the next ship in the list of ships (as the new ship $v$) and proceed from **Step IIb**. If cargo $i$ remains unassigned after all ships have been tested, remove it from $L$.

We now use the deterministic insertion heuristic to construct the rest of the solution. The insertion heuristic is a ship scheduling adaptation of the heuristic presented in [19]. First, the list of unassigned cargoes is sorted increasingly by the start of the pickup time window (time sort) or decreasingly by the cargo quantity (quantity sort). Then, the heuristic processes all cargoes in the list in sequence. Each cargo is assigned
to the available ship that gives the highest profit—given that the assignments already made are fixed. One half of the specified number of initial solutions is generated using time sort, while the other half is generated using quantity sort.

4.2. Quick local search

The quick local search explores a union of three different neighborhoods. Each neighborhood is assigned a frequency, making it possible to explore some neighborhoods more often than others. As opposed to the classic local search strategy of checking all neighbors and selecting the best, our strategy is to perform all improving passes as they are found. The search continues until a local optimum is found, i.e. when all neighborhoods are explored without finding any improvement.

The neighborhoods are a collection of intra-route and inter-route operators. Intra-route operators try to improve the schedule of one ship, while inter-route operators look for improvements by moving cargoes between two or more ships. The neighborhoods are presented in Figs. 1–3. The descriptions below do not indicate how the neighbors are enumerated; they are made as simple as possible to show how one neighbor is defined.

Fig. 1 describes the 1-resequence neighborhood. The schedule of ship \( v \) is visualized by a string of circles representing the sequence of port nodes. The dotted circles represent port nodes that are moved during the process. Here, cargo \( i \) is removed from the schedule of ship \( v \). The cargo is then re-inserted into the schedule of ship \( v \) at the best possible place. Let \( N \) be the number of cargoes in the cargo set \( \mathcal{N} \). Node \( i \) represents the loading port of cargo \( i \), while node \( N+i \) represents the unloading port of cargo \( i \). In the example shown in the figure, the loading port node is moved, while the unloading port node is re-inserted at the same place as before.

Fig. 2 illustrates the principles of the Reassign neighborhood. Cargo \( i \) is removed from the schedule of ship \( v \). Then the best insertion into each of the other ships is found. Cargo \( i \) is assigned to the ship that gives the best feasible insertion—ship \( u \). If there is one or more rejected cargoes in the cargo list, i.e.
Fig. 2. Reassign neighborhood.

Fig. 3. 2-interchange neighborhood.
cargoes for which the heuristic has not found any feasible insertion so far, the heuristic tries to find room for one of them. If a feasible reassign is found, the rejected cargo \( j \) is inserted into the schedule of ship \( v \).

Fig. 3 shows how the 2-interchange neighborhood works. Cargo \( i \) is removed from the schedule of ship \( v \), and cargo \( j \) is removed from ship \( u \). Then cargo \( i \) is inserted into the schedule of ship \( u \), while cargo \( j \) is inserted into the schedule of ship \( v \).

The quick local search heuristic is illustrated by the flowchart in Fig. 4. Only a subset of the neighborhoods is used at each iteration. The motivation behind this is that the neighborhoods have significantly different computational complexities. The idea is that complex neighborhoods should not be used as often as simple ones.

The neighborhoods are indexed by \( s \) ranging from 1 to \( S \), where \( S \) is the number of neighborhoods. At each iteration \( (iter) \) the heuristic loops through the neighborhood indices and performs one round of each

---

**Fig. 4. A flowchart of the quick local search heuristic.**
neighborhood that is allowed by the test \( f(s, \text{iter}) = \text{TRUE} \). Two integer parameters are assigned to each neighborhood:

\( FREQ_s \): inverse frequency, i.e. the number of iterations between each time neighborhood \( s \) is used.
\( FIRST_s \): first iteration neighborhood \( s \) is used.

For instance, if \( FREQ_s \) is set to 5 and \( FIRST_s \) is set to 2, the neighborhood is to be used at iteration numbers 2, 7, 12, etc.

If no improvement has been found during an iteration, the solution is locally optimal with respect to the neighborhoods that were used in that iteration. However, in order to conclude that we have a local optimum, the solution must be locally optimal with respect to all the neighborhoods in \( S \). Hence, the rest of the neighborhoods in \( S \) are used.

If no improvement is found, we have a local optimum and the quick local search heuristic terminates. One can easily imagine parameter settings where some of the iterations are “empty”, i.e. that no neighborhoods will be used in a specific iteration. Such iterations will not be performed. The heuristic also stores the neighborhoods that have been used since the last improvement was found, in order to prevent neighborhoods from being used unnecessarily.
4.3. Extended local search

The extended local search heuristic is used in the last stage of the multi-start local search heuristic to gain further improvements in the best solutions found so far. The search heuristic uses five different neighborhoods: the three neighborhoods that are used in the quick local search and two neighborhoods that are described in the following. The flowchart presented in Section 4.2 is also valid for the extended local search heuristic.

**Fig. 5** illustrates the 2-resequence neighborhood. Cargoes $i$ and $j$ are removed from the schedule of ship $v$. Cargo $i$ is then reinserted into the schedule of ship $v$ at the best possible place. Finally, cargo $j$ is reinserted at the best possible place.

**Fig. 6** presents the principles of the 3-interchange neighborhood. Cargoes $i$, $j$ and $k$ are removed from ships $v$, $u$ and $w$, respectively. The cargoes $i$, $j$ and $k$ are then inserted into the schedule of ships $u$, $w$ and $v$, respectively.

4.4. Heuristic overview

The multi-start local search heuristic uses the following additional parameters:

- **NO_INIT**: The number of initial solutions generated.
- **NO_QUICK**: The number of initial solutions to be improved by quick local search.
- **NO_EXTENDED**: The number of solutions to be improved by extended local search.
The outline of the multi-start local search heuristic is as follows:

Step I: Generate NO_INIT initial solutions.
Step II: Choose the NO_QUICK best solutions and use the quick local search heuristic to improve them.
Step III: Choose the NO_EXTENDED best solutions and use the extended local search heuristic to improve them.

5. Computational study

The computational study is performed on eight real test cases from the tramp shipping industry. The data are collected from four different shipping companies. The test cases are described in Section 5.1, while Section 5.2 describes the parameter settings that are used. The multi-start local search heuristic is tested on all cases and compared with the set partitioning approach. The results are presented in Section 5.3.

5.1. Case descriptions

Case 1 is collected from a shipping company operating in northern Europe, transporting dry bulk commodities such as rock, iron ore and aluminum. Cases 2–6 are represented by a chemical commodities shipping company operating between Europe and the Caribbean. In these cases the number of ships varies from 3 to 13. The ships come from the same fleet. Case 2 is an artificial test case based on real data. In cases 3 and 4 the rest of the fleet is engaged in fulfilling other obligations, while in cases 5 and 6 the whole fleet is available. Case 7 is collected from a shipping company that carries petroleum commodities between ports in northern Europe. Finally, case 8 is represented by a short sea shipping company transporting dry bulk commodities in northern Europe. The ships in cases 1–6 may have multiple cargoes on board simultaneously, while in cases 7 and 8 the ships can only carry full load, i.e. one cargo on board simultaneously. The case descriptions are presented in Table 1.

5.2. Heuristic parameter settings

Table 2 presents five different frequency settings for the quick and extended local search heuristics. For the quick local search, only the first three neighborhoods are used. The frequency settings are
Table 2
Frequency settings for the different neighborhoods in the local search ($FREQ_s$–$FIRST_s$)

<table>
<thead>
<tr>
<th>Parameter settings</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Neighborhood, s</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-resequence</td>
<td>1–1</td>
<td>1–1</td>
<td>1–1</td>
<td>1–1</td>
<td>1–1</td>
</tr>
<tr>
<td>Reassign</td>
<td>1–1</td>
<td>1–1</td>
<td>1–1</td>
<td>2–2</td>
<td>2–2</td>
</tr>
<tr>
<td>2-interchange</td>
<td>1–1</td>
<td>2–1</td>
<td>3–2</td>
<td>5–5</td>
<td>5–5</td>
</tr>
<tr>
<td>2-resequence</td>
<td>1–1</td>
<td>2–2</td>
<td>2–2</td>
<td>3–3</td>
<td>3–3</td>
</tr>
<tr>
<td>3-interchange</td>
<td>1–1</td>
<td>2–2</td>
<td>5–4</td>
<td>9–8</td>
<td>100–100</td>
</tr>
</tbody>
</table>

Table 3
The effect of the different frequency settings for cases 1 and 3–6

<table>
<thead>
<tr>
<th>Frequency settings</th>
<th>Best solution</th>
<th>Quickest response</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Tie between F4 and F5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Tie between all settings (F1–F5)</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

numbered from F1 to F5, and are represented by two numbers. The left number is $FREQ_s$—i.e. how many iterations that will be performed between each time the actual neighborhood $s$ is explored. The right number represents the first iteration neighborhood $s$ is used ($FIRST_s$). Frequency setting F1 represents the standard way of executing local search, exploring all neighborhoods at each iteration, while settings F2 to F4 represent an increasing emphasis on assigning low frequencies to computationally expensive neighborhoods. F5 is equal to F4, with the exception of assigning an artificially high inverse frequency (i.e. low frequency) on the triangle interchange neighborhood ($FREQ_s = 100$). For the cases presented here, this low frequency means that the triangle interchange is invoked only when no improvement can be found by any of the other neighborhoods.

In order to select a reasonable frequency setting for the multi-start local search heuristic, we tested the extended local search heuristic outside the multi-start framework. Two initial solutions were generated by using the deterministic insertion heuristic alone (i.e. $RANDOM_P = 0$). The extended local search was tested on top of these two solutions for the six multiple cargo cases. In case 2 no improvement was found by the extended local search heuristic. Table 3 shows the effect of the different frequency settings on the other five cases. Setting F3 gave the best solution in case 6. For all the other cases, there was no difference in solution value between the settings. However, the response times were different. Settings F4 and F5 gave a quicker response than the other frequency settings. It should be noted that there was a substantial effect on response times, while the effect on solution quality was negligible. These results indicate that it is computationally efficient to assign a low frequency to a complex neighborhood and use simple neighborhoods more often. On the basis of these results, frequency setting F5 was chosen for the multi-start local search heuristic.
Fig. 7. Sensitivity to the value of the parameter $\text{RANDOM}_P$ for case 5.

Fig. 7 shows how the multi-start local search heuristic results from case 5 vary with the parameter $\text{RANDOM}_P$. The smallest and average optimality gaps from ten runs are reported for each value of $\text{RANDOM}_P$. The following parameter setting was used: $\text{NO\_INIT}=100$, $\text{NO\_QUICK}=6$, $\text{NO\_EXTENDED}=1$. The figure shows that the stability of the heuristic decreases with the value of $\text{RANDOM}_P$. The “optimal” $\text{RANDOM}_P$ value, giving the smallest average optimality gap, for case 5 is 14. For the sake of brevity, we do not show charts for the other cases, but it should be noted that there are differences between the cases. The “optimal” value of $\text{RANDOM}_P$ varied from 5 in case 6 to 25 in case 4. Hence, we have chosen to set $\text{RANDOM}_P = 15$ for all cases in the following.

Two different parameter settings were tested for the multi-start local search heuristic:

- Parameter setting 1: $\text{NO\_INIT}=100$, $\text{RANDOM}_P = 15$, $\text{NO\_QUICK}=6$, $\text{NO\_EXTENDED}=1$.
- Parameter setting 2: $\text{NO\_INIT}=1000$, $\text{RANDOM}_P = 15$, $\text{NO\_QUICK}=14$, $\text{NO\_EXTENDED}=4$.

Frequency setting F5 was used in both the local search and the extended local search.

5.3. Computational results

The tests were performed on a PC with a Pentium IV, 2.8 GHz processor and 1.5 GB RAM under Windows XP. The heuristic and the a priori column generation were developed in C++ and the set partitioning model was solved by XpressMP.

In the following we will concentrate primarily on the multiple cargo cases. The full load cases represent an easier problem than the multiple cargo cases, since pickup and delivery can be modeled as one operation. However, we find it important that the heuristic works for the full load cases as well, since many shipping companies are engaged in this type of operation.

Table 4 shows the performance of the set partitioning approach for five of the six multiple cargo cases. For case 6 the set partitioning approach failed, since it was impossible to generate all columns a priori. The first two rows of the table give a good characteristic of the complexity of each case. For each combination of cargo set and ship all feasible schedules were enumerated, and the best schedule was used in the set partitioning problem. In case 4 there is on average approximately 7800 feasible schedules per cargo set—ship combination. This is a lot compared to case 2, where there are less than three feasible schedules per cargo set—ship combination. However, since the average number of cargoes per feasible schedule is 6.9 we can also compare it with an unconstrained traveling salesman problem instance with 14 cities in addition to the origin having 14! feasible schedules.
Table 4
Information from the set partitioning approach for the multiple cargo cases

<table>
<thead>
<tr>
<th>Cases</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set partitioning information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of feasible schedules</td>
<td>1306704</td>
<td>327</td>
<td>2609685</td>
<td>26512819</td>
<td>19240874</td>
</tr>
<tr>
<td># of cargo set—ship combinations</td>
<td>46254</td>
<td>113</td>
<td>6973</td>
<td>3399</td>
<td>21257</td>
</tr>
<tr>
<td>Avg. # of cargoes per feasible schedule</td>
<td>6.0</td>
<td>4.1</td>
<td>7.0</td>
<td>7.6</td>
<td>6.9</td>
</tr>
<tr>
<td>Max # of cargoes in a feasible schedule</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Column generation CPU (s)</td>
<td>6482</td>
<td>&lt;1</td>
<td>9300</td>
<td>13325</td>
<td>27469</td>
</tr>
<tr>
<td>Set partitioning CPU (s)</td>
<td>5</td>
<td>&lt;1</td>
<td>&lt;1</td>
<td>&lt;1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5
Solution qualities and computational results from testing the multi-start local search heuristic

<table>
<thead>
<tr>
<th>Cases</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter setting 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest gap from best solution (%)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>Average gap (%)</td>
<td>0.1</td>
<td>0.0</td>
<td>0.5</td>
<td>1.3</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>Average heuristic CPU (s)</td>
<td>4</td>
<td>0.5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>72</td>
</tr>
<tr>
<td>Parameter setting 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest gap from best solution (%)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>Average gap (%)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.7</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>Average heuristic CPU (s)</td>
<td>22</td>
<td>4</td>
<td>17</td>
<td>16</td>
<td>30</td>
<td>283</td>
</tr>
</tbody>
</table>

From Table 4 it can be seen that it is a computationally expensive task to generate all columns a priori even for relatively small problem instances. Only for case 2, that is very tightly constrained, the set partitioning approach proved to be highly efficient. It is interesting to note that when all columns were generated, the corresponding set partitioning problem was easy to solve. The optimal solution was found in the first branch-and-bound node for all the five cases.

Table 5 shows the results from testing the multi-start local search heuristic. For cases 1–5 the heuristic solutions are compared with the optimal solution. We cannot say anything about the optimality gap for case 6, so for this case we use the best heuristic result (using any setting) for comparison. Since the heuristic is stochastic regarding the generation of initial solutions, we performed ten runs in order to make sensible comparisons. The smallest gap and the average gap from the best solution and the average response time are reported for parameter settings 1 and 2.

For cases 1 to 3 the heuristic produces excellent results already for parameter setting 1 with 100 initial solutions. It should be noted that for case 2 the optimal solution was found among the initial solutions in all runs. For parameter setting 2, with 1000 initial solutions, the multi-start local search heuristic produced near-optimal solutions in cases 4 and 5 as well, and the performance was stable for all the multiple cargo cases. The results show that when using this heuristic one has to make some compromise between solution
Table 6
Computational results from testing the set partitioning approach and the heuristic—full load cases

<table>
<thead>
<tr>
<th>Cases</th>
<th>Set partitioning approach</th>
<th>Multi-start local search heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of feasible schedules</td>
<td>62 877</td>
</tr>
<tr>
<td></td>
<td># of cargo combinations</td>
<td>31 330</td>
</tr>
<tr>
<td></td>
<td>Avg. # of cargoes per feasible schedule</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>Max # of cargoes in a feasible schedule</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Column generation CPU (s)</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>Set partitioning CPU (s)</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td># of branch-and-bound nodes</td>
<td>5101</td>
</tr>
<tr>
<td></td>
<td>Smallest gap from optimal solution (%)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Average gap (%)</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>Average response time (s)</td>
<td>65</td>
</tr>
</tbody>
</table>

quality and response time. For instance, for case 5, if you use parameter setting 1, you can expect to obtain a solution that is less than 1% from the optimal solution in 4 s. With parameter setting 2 the average gap is 0.4% with an average response time of 30 s. And, finally, if you choose to run the heuristic ten times with parameter setting 2, you can expect a gap of approximately 0.1% in 300 s.

The difference in response time between the set partitioning approach and the heuristic is very large; see Tables 4 and 5. The set partitioning approach solves case 1 to optimality with more than 100 min of computation time, while the heuristic actually finds the optimal solution in many of the runs within a CPU time of 4 s. A similar situation is seen for case 3.

Table 6 shows computational results for the full load cargo cases. The heuristic was run ten times with parameter setting 2. The table shows that the heuristic works well also for these cases. For both cases the optimality gap is small. The set partitioning approach was very efficient for these cases. This is probably because they are more constrained than the multiple cargo cases, and that the pickup and delivery of each cargo can be treated as one operation. It is interesting to observe that case 7 was—as the only case in this paper—not solved in the first node of the branch-and-bound tree.

6. Conclusions

We have presented a multi-start local search heuristic for solving tramp ship scheduling problems. A set partitioning approach where all columns were generated a priori was used as a benchmark.

In the multi-start local search heuristic, a set of initial solutions is constructed by a partly randomized insertion heuristic. A local search heuristic is then used to improve a given number of the best initial solutions. In order to increase efficiency, the local search is divided into a quick and an extended version. Both local search heuristics explore a collection of neighborhoods with different frequencies. The extended local search heuristic uses a larger collection of neighborhoods than the quick version. The quick local search heuristic is used to improve all the selected initial solutions, while the extended version is used to
improve a small number of the quick local optima. A version of the multi-start local search heuristic is
implemented in the commercial decision support system TurboRouter [20], which is applied by several
shipping companies.

In our opinion, the multi-start local search heuristic is well suited for the tramp ship scheduling problem
since real-life instances are often tightly constrained. The computational study supports this and shows
that the heuristic provides optimal or near-optimal solutions for real-life instances within a reasonable
response time. The heuristic also shows good robustness in terms of solution quality. Robust results in a
short response time may often be crucial for planners in the shipping industry, for instance giving quick
answers to customers about whether to accept or reject spot cargoes. The results indicate that a decision
support system based on the described heuristic will be a valuable tool for planners in the shipping
industry.

There are several interesting future research areas for tramp ship scheduling problems. A Dantzig–Wolfe
decomposition approach would be able to solve larger cases than the set partitioning approach with a
priori column generation. It would also be interesting to investigate other types of heuristics, such as
a tailormade tabu search heuristic. Benchmark test cases are demanded. Such cases would probably
accelerate the development of high quality solution methods, like we have seen within vehicle routing
research. Furthermore, it would be interesting to consider richer models of the problem. As an example,
we can mention that in many contracts between shipping companies and cargo owners the quantity of
each cargo is not fixed, but is rather given in an interval. The introduction of soft time windows is another
example; see [21] and [22] for some particular ship scheduling problems with soft time windows. In
practice, time windows are seldom seen as “hard” constraints. In the modeling, this might be handled by
enforcing a penalty each time a time window is broken. Introducing such features into the model provides
new challenges in the development of solution methods.

Acknowledgements

This work was carried out with financial support from the TOP project on transport optimization and
the Research Council of Norway.

References

1989;36:27–42.


