Tramp Ship Routing and Scheduling with Voyage Separation Requirements

Vilhelmsen, Charlotte

Publication date:
2014

Document Version
Peer reviewed version

Tramp Ship Routing and Scheduling with Voyage Separation Requirements

Charlotte Vilhelmsen Richard M. Lusby

Department of Management Engineering, Technical University of Denmark
Produktionstorvet, Building 426, 2800 Kgs. Lyngby, Denmark
chaan@dtu.dk, rmlu@dtu.dk

August 1, 2014

Abstract

In this paper we explore tramp ship routing and scheduling. Tramp ships operate much like taxies following the available demand as opposed to liner ships that operate more like busses on a fixed route network according to a published timetable. Tramp operators can determine some of their demand in advance by entering into long term contracts and then try to maximise profits from optional voyages found in the spot market. Routing and scheduling a tramp fleet to best utilise fleet capacity according to current demand is therefore an ongoing and complicated problem. Here we add further complexity to the routing and scheduling problem by incorporating voyage separation requirements that enforce a minimum time spread between some voyages. The incorporation of these separation requirements helps balance the conflicting objectives of maximising profit for the tramp operator and minimising inventory costs for the charterer, since these costs increase if similar voyages are not performed with some separation in time. We have developed a new and exact Branch-and-Price procedure for this problem. We use a dynamic programming algorithm to generate columns and use a modified time window branching scheme to enforce the voyage separation requirements which we relax in the master problem. Computational results show that our algorithm finds very good if not optimal solutions extremely fast though one instance requires longer time. We also compare our method to an earlier a priori path generation method which we show is not optimal. Computational results confirm this as our algorithm consistently finds solutions that are equally good or better than those from the a priori generation method. Furthermore, for all but one instance, our solutions are found in the same or shorter time than those from the a priori generation method.

1 Introduction

With over 9 billion tons of cargo transported by the international shipping industry every year (UNCTAD, 2013) there is no doubt that the maritime sector plays a vital role in world trade. Maritime transportation therefore constitutes an important research area and in this paper we explore tramp shipping where ships sail much like taxies following the available demand. This is in contrast to liner shipping where ships operate more like busses following a fixed route network and a published timetable. Tramp operators do however know some of their demand in advance since they can enter into long term contracts and then seek to maximise profit from optional cargoes found in the spot market.

For tramp operators a very important an ongoing problem is how to most efficiently route and schedule their fleets according to current demand. This is precisely the problem we consider here. However, we add further complexity to the problem by incorporating voyage separation requirements which enforce a minimum time spread between some voyages. This is done in order to find a balance between maximising profit for the ship operator and increasing customer satisfaction for the charterer who faces increased inventory costs if similar voyages are not performed with some separation in time. Thereby, this incorporation of voyage separation requirements correspond to a crude way of viewing the tramp ship routing and scheduling problem in the broader context of the supply chain. Furthermore, with already low freight rates and increased competition amongst operators, the customer satisfaction perspective is interesting and Norstad et al. (2013) show that it can be incorporated with only a small decrease in profit. Results from Norstad et al. (2013) show that their methods struggle on larger and more complex problem instances and so, the aim here is to develop a more efficient solution method.

In this paper we therefore consider the tramp ship routing and scheduling problem with voyage separation requirements. We present a mixed integer programming formulation for this problem and devise a new, exact solution method for it. This solution method is a Branch-and-Price procedure with a dynamic programming algorithm to generate the columns. A modified time window branching scheme is used to enforce the voyage separation requirements, which are relaxed in the master problem. Computational results on 16 instances show that this method finds very good if not optimal solutions extremely fast although one instance requires longer time. We also compare our algorithm to an a priori path generation method from Norstad et al. (2013). We argue that their method is not optimal and computational results verify this. In fact, the profits from our solutions are consistently equal to or better than theirs and are as much as 6% higher on one instance. Furthermore, on all but one instance our solution is obtained in the same or shorter time than that of the a priori path generation method.

The remainder of the paper is organized as follows. In Section 2 relevant literature is presented while Section 3 provides a problem description as well as a non-linear mathematical model for the problem. Section 4 describes the decomposition of the problem and the dynamic column generation procedure, i.e. the pricing part of the proposed algorithm. Section 5 describes the branching part of the algorithm, namely time window branching and constraint branching. In Section 6, we describe the data instances used to evaluate the performance of our algorithm as well as the results obtained with our algorithm on these instances. We also compare these results to results obtained from running the a priori path generation method from Norstad et al. (2013) on these same instances. Finally, concluding remarks and suggestions for future work are discussed in Section 7.

2 Literature Review

Mathematical formulations and discussions on solution methods for a wide range of maritime problems on all planning levels can be found in Christiansen et al. (2007). Furthermore, a thorough review of literature focused on ship routing and scheduling before 2013 can be found in the four review papers, Ronen (1983), Ronen (1993), Christiansen et al. (2004) and Christiansen et al. (2013). The tramp ship routing and scheduling problem considered here is also closely related to the vehicle routing problem with time windows, for which we refer the reader to Cordeau et al. (2002).

Recent work on tramp ship routing and scheduling include Cóccola et al. (2014) who present a novel column generation approach in which the conventional dynamic programming route-generator is replaced by a continuous time MILP slave problem. Their computational results show that their method outperforms both a pure exact optimisation model and a heuristic solution method previously reported in the literature. Castillo-Villar et al. (2014) present a Variable Neighborhood Search based heuristic procedure for solving the tramp ship routing and scheduling problem with discretised time windows. They ignore optional cargoes and therefore seek to minimise cost rather than to maximise profit. However, the discretisation approach allows them to incorporate several practical extensions and they specifically investigate the inclusion of variable speed. Stålhane et al. (2014) combine traditional tramp shipping with a vendor managed inventory service in an

attempt to challenge the traditional *Contract of Affreightment* which is the standard agreement between a tramp ship company and a charterer. They present an exact solution method as well as a heuristic for solving larger instances. Stålhane et al. (2012) and Vilhelmsen et al. (2013) both extend the basic problem by investigating, respectively, split loads and the integration of bunker planning. As further extensions to the basic problem, Fagerholt and Ronen (2013) present and consolidate results for three practical extensions within bulk shipping: (1) flexible cargo quantities, (2) split cargoes, and (3) sailing speed optimization. Kang et al. (2012) consider the interaction between ship routing and scheduling and ship deployment, though in a context quite different from ours. Somewhere between tramp shipping and liner shipping, Moon et al. (2014) also investigate a combined ship routing and fleet deployment problem.

In this paper we extend the basic tramp ship routing and scheduling problem by including voyage separation requirements. Such requirements are also considered in Norstad et al. (2013) in a context similar to ours. Their computational results show that the introduction of voyage separation requirements only leads to a marginal reduction in profit. They present an arc flow formulation as well as a path flow formulation solved using a priori path generation. They conclude that both models work well on small problem instances and that the path flow model is also capable of solving real life size instances within an acceptable time. However, for larger instances, or situations with further complexity, neither of the two solution methods are applicable. We investigate the path flow approach further in Section 6, where we also compare it to our solution method.

Within liner shipping, another example of time separation constraints can be found in Sigurd et al. (2005). They consider a variant of the general pickup and delivery problem with multiple time windows and the addition of requirements for recurring visits, separation between visits and limits on transport lead-time. They use a heuristic branch-and-price algorithm to obtain a fixed visit schedule with a recurring route for each ship.

Within other transportation modes, we can find numerous examples of time separation requirements. Especially synchronisation constraints are often encountered in vehicle routing, see e.g. Reinhardt et al. (2013) who consider synchronized dial-a-ride transportation for airport passengers with reduced mobility or Drexl (2013) who extend the vehicle routing problem to include trailers and transshipments and describe how to model several important problems within this context. More general temporal dependencies are handled in Dohn et al. (2011) for the vehicle routing problem with time windows. They present a comprehensive description of literature dealing with both synchronisation constraints and more general temporal dependencies within various transport modes, crew scheduling and even machine scheduling. They present two compact formulations and the Dantzig-Wolfe decompositions of these formulations. Four different master problem formulations are proposed along with a time window branching scheme used to enforce feasibility on the relaxed master problems. Their computational study shows that, depending on the problem at hand, the best performance is achieved either by relaxing the temporal dependency constraints in the master problem, or by using a time-indexed model, where generalized precedence constraints are added as cuts when they become severely violated.

3 Problem Description

In this section we give a problem description and present a mathematical formulation for the Tramp Ship Routing And Scheduling Problem with Voyage Separation Requirements (TSRSPVSR).

A tramp operator typically has long term contracts that obligate him to perform some voyages; however, he/she can choose to perform additional voyages, so called *spot voyages*, if fleet capacity allows it and it is profitable. The objective is to create a profit maximizing set of fleet schedules, one for each ship in the fleet, where a schedule is a sequence and timing of port calls representing the voyages. The optimal solution therefore combines interdependent decisions on which optional voyages to perform, the assignment of voyages to ships and the optimal sequence and timing of port calls for each ship. If capacity is insufficient to perform all mandatory contract voyages, it is possible to charter in spot vessels to perform some of these.

A voyage is mainly characterized by the quantity to be transported, the revenue obtained from transporting it and the pickup and discharge ports. There is also a ship specific service time in ports and a time window giving the earliest and latest start for each voyage. We assume that a ship can only perform one voyage at a time corresponding to the case of full shiploads.

Several voyages can be identical except for their time windows for start of service. In fact, contract cargoes stem from contracts of affreightment and these often state that the operator must perform a specific voyage a given number of times during a predefined time interval, e.g. three times during a month. Since such voyages correspond to the same geographical route, we group them according to these *trade routes*. Thereby, the tramp operator has contract trade routes on which a specific number of identical voyages must be performed and can choose to perform additional spot voyages. Spot voyages can also be grouped according to trade routes. However, any number of voyages on such spot trade routes can be performed as opposed to contract trade routes where all voyages must be performed.

Contracts of affreightment often contain a contract clause stating that voyages must be performed 'fairly evenly spread' in time without specifically defining what this means. In practice it means that, following the previous example with a contract trade requiring three voyages during a month, the tramp operator should not perform all three voyages within the first week and then do nothing for the remaining three weeks. As discussed in Norstad et al. (2013), these contract clauses can be handled either by imposing additional time windows on voyage start times or by imposing restrictions on the minimum time spread between the start of consecutive voyages on the same trade. Obviously, there is a trade off between the quality of service provided to the charterer and the flexibility of the tramp operator. Using restrictions on the minimum time spread seems to provide the best balance between these two conflicting objectives, and so we adhere to this option here, just as in Norstad et al. (2013). Note that this time spread must be adhered to even if spot vessels are involved.

A tramp fleet is usually heterogeneous, comprised of ships of different sizes, load capacities, fuel consumptions, speeds, and other characteristics. Ships can be occupied with prior tasks when planning starts so each ship is further characterized by the time it is available for service and the location it is at when it becomes available. There are also maintenance requirements for some ships and these must be respected in the scheduling process. The characteristics of a ship determine which voyages it is compatible with.

As we consider a fixed fleet we can disregard the fixed setup costs and focus on the variable operating costs. The main sailing cost is fuel cost and this is different for each ship and depends on both its speed and its load. Since we assume full shiploads, we can factor in load dependency by simply using two fuel consumption functions: One for ballast legs and one for laden legs. Each ship is assumed to sail at two predefined speed settings: one for ballast legs and one for laden legs and so, the two fuel consumption functions are in effect two constants used for the two types of sailing legs. When loading and discharging, ship dependent port costs are incurred. Finally, there is a cost associated with chartering in spot vessels to perform uncovered contract voyages.

3.1 Mathematical model

Let $\mathcal{V}$ be the set of ships. Furthermore, let $\mathcal{R}$ denote the set of trade routes and associate with each trade route $r \in \mathcal{R}$ the set of voyages $\mathcal{I}_r = \{1, 2, \ldots, n_r\}$ on trade route r during the planning horizon. A specific voyage $i \in \mathcal{I}_r$ for $r \in \mathcal{R}$ can be denoted by the pair (r, i). Thereby, the two pairs (r, i) and $(r, i + 1)$ denote two consecutive voyages on trade r. Spot voyages and maintenance requirements are also modeled using this trade route notation though n_r is equal to 1 for maintenance trades.

Due to port and cargo compatibility, capacity requirements and other restrictions, not all ships can sail all trade routes. Therefore, we further define $\mathcal{R}^k$ and $\mathcal{V}_r$ as, respectively, the set of trade routes compatible with ship $k \in \mathcal{V}$ and the set of ships compatible with trade route r. We let $\mathcal{N}_C$, $\mathcal{N}_O$ and $\mathcal{N}_M^k$ denote, respectively, the set of contract voyages, the set of optional voyages and the set of maintenance requirements for ship k.

In order to define the problem on a graph, we define an origin and a destination node for each ship $k \in \mathcal{V}$ and denote these $o(k)$ and $d(k)$ respectively. The origin node corresponds to the location of the ship when planning starts while the destination node is artificial and simply corresponds to the geographical location of ship k at the end of the planning horizon.

The problem can then be defined on the graph $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \mathcal{N}_C \cup \mathcal{N}_O \cup_{k \in \mathcal{V}} \mathcal{N}_M^k \cup_{k \in \mathcal{V}} \{o(k), d(k)\}$. If a voyage or maintenance, (r, i), can be performed directly before another voyage or maintenance, (q, j), then $\mathcal{A}$ contains the arc $((r, i), (q, j))$. $\mathcal{A}$ also contains the arcs $(o(k), (r, i))$ and $((r, i), d(k))$ for each ship k that can perform voyage i on trade route r. Finally, for each ship $k \in \mathcal{V}$ without maintenance requirements, the set also contains the arc $(o(k), d(k))$ corresponding to an idle ship. For each ship $k \in \mathcal{V}$ we further define the set $\mathcal{A}^k$ as the set of arcs in $\mathcal{A}$ that are traversable by ship k, e.g. with respect to time.

For each node $(r, i) \in \mathcal{N}$ we have a time window $[a_{ri}, b_{ri}]$ describing the earliest and latest time to start service for the corresponding voyage or maintenance. For $o(k)$ this window is collapsed into the time that ship k is available for service, $a_{o(k)}$. We let B_r denote the minimum acceptable time between the start of service on two consecutive voyages on trade route $r \in \mathcal{R}$.

We use T^k_{riqj} to denote the fixed time for performing voyage or maintenance $(r, i) \in \mathcal{N}$ and sailing ballast to the first pickup port of voyage or maintenance $(q, j) \in \mathcal{N}$ with ship k. T^k_{riqj} includes service time in ports for voyage/maintenance (r, i), laden travel time for voyage legs and ballast travel time from the final discharge port of voyage/maintenance (r, i) to the first pickup port of voyage/maintenance (q, j). Similarly, we let $T^k_{o(k)ri}$ denote the time for traveling ballast with ship k from its origin to the first port of voyage or maintenance (r, i).

For the ballast legs, C^k_{riqj} and $C^k_{o(k)ri}$ denote, respectively, the cost of traveling ballast with ship k from the last discharge port of voyage or maintenance $(r, i) \in \mathcal{N}$ to the first pickup port of voyage or maintenance $(q, j) \in \mathcal{N}$, and from the origin node to the first pickup port of voyage or maintenance (r, i). We also have ship specific profits for the laden voyage legs. These profits, P^k_{ri}, take into account the revenue incurred from performing voyage (r, i), the cost of sailing laden with ship k from the first pickup port of the voyage to the final discharge port of the voyage, and finally, the port costs incurred during the voyage or maintenance period when performed by ship k. If a voyage (r, i) is instead performed by a spot vessel, the cost incurred is C^S_{ri}.

For the mathematical formulation we need several variables. First, we define binary flow variables x^k_{riqj} for $k \in \mathcal{V}$, $((r, i)(q, j)) \in \mathcal{A}^k$ that are equal to 1, if ship k performs voyage (r, i) just before voyage (q, j), and 0 otherwise. Likewise, we define binary flow variables $x^k_{o(k)ri}$, $x^k_{rid(k)}$ and $x^k_{o(k)d(k)}$ for the arcs connecting the origin and destination nodes with other nodes and with each other. The start time for service at each node is also variable and so we define time variables $t^k_{o(k)}$ and t^k_{ri} for each $k \in \mathcal{V}$, $r \in \mathcal{R}^k$ and $i \in \mathcal{I}_r$. If a spot vessel is hired to service a contract voyage $(r, i) \in \mathcal{N}_C$, we denote the start time by t^S_{ri} and let the binary variable y_{ri} be equal to 1.

We can now give an arc flow formulation of the TSRSPVSR:

$$\max \sum_{k\in\mathcal{V}} \sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} \sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} (P^k_{ri} - C^k_{riqj}) x^k_{riqj} + \sum_{k\in\mathcal{V}} \sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} P^k_{ri} x^k_{rid(k)} - \sum_{k\in\mathcal{V}} \sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} C^k_{o(k)ri} x^k_{o(k)ri} - \sum_{(r,i)\in\mathcal{N}_C} C^S_{ri} y_{ri} \tag{1}$$

s.t.

$$\sum_{k\in\mathcal{V}_r} \Big(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)} \Big) + y_{ri} = 1, \qquad \forall (r,i)\in\mathcal{N}_C, \tag{2}$$

$$\sum_{k\in\mathcal{V}_r} \Big(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)} \Big) \le 1, \qquad \forall (r,i)\in\mathcal{N}_O, \tag{3}$$

$$\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)} = 1, \qquad \forall k\in\mathcal{V},\ (r,i)\in\mathcal{N}^k_M, \tag{4}$$

$$\sum_{r\in\mathcal{R}_k} \sum_{i\in\mathcal{I}_r} x^k_{o(k)ri} + x^k_{o(k)d(k)} = 1, \qquad \forall k\in\mathcal{V}, \tag{5}$$

$$x^k_{o(k)ri} + \sum_{q\in\mathcal{R}_k} \sum_{j\in\mathcal{I}_q} x^k_{qjri} - \sum_{q\in\mathcal{R}_k} \sum_{j\in\mathcal{I}_q} x^k_{riqj} - x^k_{rid(k)} = 0, \qquad \forall k\in\mathcal{V},\ r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{6}$$

$$\sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} x^k_{rid(k)} + x^k_{o(k)d(k)} = 1, \qquad \forall k\in\mathcal{V}, \tag{7}$$

$$x^k_{o(k)qj}(t^k_{o(k)} + T^k_{o(k)qj} - t^k_{qj}) \le 0, \qquad \forall k\in\mathcal{V},\ q\in\mathcal{R}^k,\ j\in\mathcal{I}_q, \tag{8}$$

$$x^k_{riqj}(t^k_{ri} + T^k_{riqj} - t^k_{qj}) \le 0, \qquad \forall k\in\mathcal{V},\ ((r,i),(q,j))\in\mathcal{A}^k, \tag{9}$$

$$a_{ri}\Big(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)} \Big) \le t^k_{ri} \le b_{ri}\Big(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)} \Big), \qquad \forall k\in\mathcal{V}, r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{10}$$

$$\sum_{k\in\mathcal{V}_r} t^k_{ri} + t^S_{ri} y_{ri} + B_r \le \sum_{k\in\mathcal{V}_r} t^k_{r,i+1} + t^S_{r,i+1} y_{r,i+1}, \qquad \forall (r,i)\in\mathcal{N}_C, i\in\mathcal{I}_r\setminus\{n_r\}, \tag{11}$$

$$a_{ri} \le t^S_{ri} \le b_{ri}, \qquad \forall (r,i)\in\mathcal{N}_C, \tag{12}$$

$$t^k_{o(k)} \ge a_{o(k)}, \qquad \forall k\in\mathcal{V}, \tag{13}$$

$$x^k_{o(k)ri}\in\{0,1\}, \qquad \forall k\in\mathcal{V},\ r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{14}$$

$$x^k_{o(k)d(k)}\in\{0,1\}, \qquad \forall k\in\mathcal{V} : \mathcal{N}^k_M = \emptyset, \tag{15}$$

$$x^k_{riqj}\in\{0,1\}, \qquad \forall k\in\mathcal{V},\ ((r,i),(q,j))\in\mathcal{A}^k, \tag{16}$$

$$x^k_{rid(k)}\in\{0,1\}, \qquad \forall k\in\mathcal{V},\ r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{17}$$

$$y_{ri}\in\{0,1\}, \qquad \forall (r,i)\in\mathcal{N}_C. \tag{18}$$

The objective function (1) maximises profit by subtracting all spot vessel costs and ballast leg costs from profits obtained on laden voyage legs performed by ships in the fleet. Constraints (2) and (3) ensure that all contract voyages are performed by exactly one ship, possibly a spot vessel, and that all spot voyages are performed by at most one ship. For ships with maintenance requirements, constraints (4) ensure that these requirements are adhered to. Constraints (5) and (7) together with the flow conservation constraints in (6) ensure that each ship is assigned a schedule starting at the origin node and ending at the destination node. Constraints (8) ensure that if the schedule for ship k visits node $o(k)$ directly before node (q,j), the service at node (q,j) cannot begin before service time at node $o(k)$ plus travel time from node $o(k)$ to node (q,j) with ship k. Since waiting time is allowed, the constraints have an inequality sign. Constraints (9) impose similar restrictions when the starting node is a voyage or maintenance node (r,i). In such cases, port time at node

(r,i) plus travel time for performing voyage (r,i) must also be accounted for before service at node (q,j) can start. In the time window constraints (10), the service time for ship k at node (r,i), t^k_{ri}, is forced to zero if ship k does not visit node (r,i). For all consecutive voyage pairs, (r,i) and $(r,i+1)$, constraints (11) ensure that the time spread between start of service for the two voyages is at least as large as the required time spread, B_r, on trade route r. Note that in order to also enforce the time spread when voyages are performed by spot vessels, we must ensure that spot vessel visits also adhere to the time windows and this is taken care of in constraints (12).

Constraints (13) ensure that no ship can start its schedule before it is available for service. The flow variables are restricted to be binary in (14)-(17) while constraints (18) impose similar restrictions on the spot vessel decision variables. Note that due to constraints (2) and the binary restrictions on the flow variables, we do not actually need the binary restrictions on the spot vessel variables. However, we include them for completeness sake and also to exploit their binary nature in the branch-and-bound scheme later.

4 Decomposition

The nonlinear mixed integer programming model (1)-(18) could in theory be solved by commercial optimization software for linear problems after linearising constraints (8), (9), and (11). However, as pointed out in Norstad et al. (2013), where they use a similar arc flow model, most real life problem instances will be too large to achieve solutions in a reasonable amount of time. This section therefore describes a solution method tailored for the TSRSPVSR.

In the mathematical programming model (1)-(18), constraints (4)-(10) and (13)-(17) are ship specific with no interaction between ships. They constitute a routing and scheduling problem for each ship where maintenance and time windows are considered. The objective function also splits into separate terms for each ship, aside from the last part corresponding to the cost of using spot vessels. The only constraints linking the ships together are the common constraints (2), (3) and (11) which ensure that each contract cargo is carried by exactly one ship, that each spot cargo is carried by at most one ship, and that the voyage separation requirements (VSR) are fulfilled. This suggests use of decomposition and column generation since it allows the complex and ship specific constraints, concerning the routing and scheduling, to be handled separately in subproblems, one for each ship. Only the common constraints and the spot vessel constraints (12) and (18) remain in the master problem in which feasible ship schedules constitute the columns. This way the original problem is transformed into a master problem with a reduced number of constraints but with a potentially very large number of columns.

If a schedule contains waiting time, we can redistribute the waiting time and thereby obtain a different schedule corresponding to the same ship and geographical route. Thereby, each geographical route can correspond to numerous different feasible schedules all with the same profit. Without the VSR constraints, it would only be necessary to include one of these schedules in the master problem while the rest could be discarded. Furthermore, for each ship, if the same voyage and maintenance stops could be ordered into numerous different geographical routes and hence, even further different schedules, only the schedule with the highest profit would have to be included in the master problem. However, due to the VSR constraints, the actual timing of port calls in a schedule for one ship can affect the timing of port calls for schedules of other ships. Therefore, any feasible schedule for a ship must be considered a valuable contribution to the master problem and so, the master problem column set can contain several schedules all corresponding to the same set of voyage and maintenance stops and even to the same geographical route. Thereby, a priori generating all columns will obviously be very time consuming and also result in very large master problems. Therefore, we turn to dynamic column generation (see e.g. Desaulniers et al. (2005) for a general description or Christiansen et al. (2007) for a maritime version) where new master problem columns that have the potential to improve the current solution are iteratively generated by the subproblems. With a relaxation of the master problem the entire solution process is therefore a Branch-and-Price procedure where new columns are iteratively priced out at each node of the search tree guided by dual variables from the current solution to the master problem.

4.1 Master Problem

The common constraints (2), (3) and (11) in combination with the spot vessel constraints (12) and (18) and the objective function (1) constitute the master problem. They must, however, be expressed by new path flow variables corresponding to feasible ship schedules and constraints must be added to ensure that each ship is assigned exactly one schedule. We let $\mathcal{S}^k$ denote the set of all feasible schedules for ship k.

Note that rather than selecting exactly one distinct schedule for each ship, we can also allow convex combinations of different schedules for each ship, as long as the chosen schedules correspond to the same geographical route. We therefore denote the set of geographical routes for ship k by $\mathcal{G}^k$. Furthermore, we expand notation on the schedule sets so that now $\mathcal{S}^k_g$ denotes the set of all feasible schedules for ship $k \in \mathcal{V}$ on geographical route $g \in \mathcal{G}^k$.

We denote the profit of a schedule by p^k_s for $k \in \mathcal{V}$, $g \in \mathcal{G}^k$, $s \in \mathcal{S}^k_g$, and define a binary schedule variable λ^k_s that is equal to 1 if ship k is chosen to sail schedule s, and 0 otherwise. The profit p^k_s is calculated based on information from the underlying schedule, which holds all necessary information, i.e. the ship it is constructed for, the voyages conducted, and the timing of port calls during the schedule. We reuse the definition of y_{ri} from the arc flow formulation and let A^k_{ris} be equal to 1 if ship k performs voyage or maintenance stop (r,i) in schedule s, and 0 otherwise. Finally, we denote the start time for voyage or maintenance node (r,i) in schedule s with ship k by T^k_{ris}. Note that these are determined in the subproblems and are therefore constants in the master problem.

The master problem can now be stated as the following path flow reformulation of the original arc flow model:

$$\max \sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}^k_g} p^k_s \lambda^k_s - \sum_{(r,i)\in\mathcal{N}_C} C^S_{ri} y_{ri} \tag{19}$$

s.t.

$$\sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}^k_g} A^k_{ris} \lambda^k_s + y_{ri} = 1, \qquad \forall (r,i) \in \mathcal{N}_C, \tag{20}$$

$$\sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}^k_g} A^k_{ris} \lambda^k_s \leq 1, \qquad \forall (r,i) \in \mathcal{N}_O, \tag{21}$$

$$\sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}^k_g} \lambda^k_s = 1, \qquad \forall k \in \mathcal{V}, \tag{22}$$

$$\sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}^k_g} T^k_{ris} \lambda^k_s + t^S_{ri} y_{ri} + B_r \leq \sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}^k_g} T^k_{r,i+1,s} \lambda^k_s + t^S_{r,i+1} y_{r,i+1}, \quad \forall (r,i) \in \mathcal{N}_C, i \in \mathcal{I}_r \setminus \{n_r\}, \tag{23}$$

$$a_{ri} \leq t^S_{ri} \leq b_{ri}, \qquad \forall (r,i) \in \mathcal{N}_C, \tag{24}$$

$$\sum_{s\in\mathcal{S}^k_g} \lambda^k_s \in \{0,1\}, \qquad \forall k \in \mathcal{V},\, g \in \mathcal{G}^k, \tag{25}$$

$$y_{ri} \in \{0,1\}, \qquad \forall (r,i) \in \mathcal{N}_C. \tag{26}$$

To solve the problem in an LP based branch-and-price framework, we first relax the binary constraints on the decision variables. The VSR constraints (23) will complicate the subproblems as the dual variables of these constraints will create linear node costs in the subproblems. Furthermore, as T^k_{ris} is a non-binary parameter, the presence of the VSR constraints in the master problem will most likely lead to more fractional solutions as it compromises the strong integer properties of the constraint matrix (we return to this matter in Section 5.2). Therefore, we also relax the VSR constraints (23) and will instead handle these when branching. Due to this relaxation of the VSR constraints, we no longer need the time window restrictions on the spot vessel time variables in (24) and so we also relax these.

As already mentioned, we use dynamic column generation to solve the problem. Therefore, we initially consider only a subset of the master problem columns, i.e. a subset of the feasible

schedules for each ship, and iteratively generate new columns that have the potential to improve the current solution. The entire branch-and-price process therefore begins with the solution of the *restricted master problem* (RMP) which is the linear relaxation of the problem (19)-(22) but with only a subset of the columns included. The dual variables from this solution are then used in the subproblems, also called pricing problems, to generate new promising columns that are added to the RMP. This iterative process continues until no further columns can improve the current master problem solution. If the current solution to the RMP is infeasible with respect to either or both the relaxed integrality and VSR constraints, we branch and continue this entire process by pricing out new columns at each node of the search tree until a feasible, optimal solution is obtained, or a specified time limit elapses.

Initially we only include a small number of feasible schedules in the RMP. To ensure that each contract cargo can actually be carried, we include a spot vessel schedule for each of the contract cargoes. For each ship in the fleet we include a schedule containing only required maintenance stops corresponding to the ship not performing any voyages for the entire planning horizon.

4.2 Subproblems - Pricing out new schedules

Constraints (4)-(10) and (13)-(17) split into one independent subproblem for each ship. Since these are all essentially the same problem, we simply consider the generic subproblem for ship k and refer to 'the subproblem'. Note that there is interdependence between the subproblems due to the common constraints. The ship routing constraints in the subproblem ensure that any solution is a feasible schedule for ship k and the objective must ensure that only schedules with the potential to improve the current solution of the RMP are generated. This means finding schedules with positive reduced costs in the current solution of the RMP.

Let π_{ri} be the dual variables for constraints (20) and (21) where the variables corresponding to (20) are free of sign while the variables corresponding to (21) must be nonnegative. Next, define $\pi_{ri} = 0$ for all $(r, i) \in \mathcal{N}_M^k$ and let ω_k be the dual for constraint (22) which is also free of sign. Since we consider the generic subproblem we can drop the superscript k on the variables and the subproblem is then given by:

$$\begin{aligned}\max \sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r}\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q}\big(P^k_{ri} - C^k_{riqj} - \pi_{ri}\big)x_{riqj} \\ + \sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r}\big(P^k_{ri} - \pi_{ri}\big)x_{rid(k)} \\ - \sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r} C^k_{o(k)ri}x_{o(k)ri} - \omega_k \qquad (27)\end{aligned}$$

s.t.

$$(4) - (10) \text{ and } (13) - (17). \qquad (28)$$

The subproblem finds the maximum reduced cost feasible schedule with respect to the current dual values. If this schedule has a positive reduced cost, it has the potential to improve the current solution to the RMP. The subproblem can be modeled as an elementary shortest path problem with resource constraints (ESPPRC) which is $\mathcal{NP}$-hard in the strong sense (see Dror (1994)).

The collaborating tramp operator is involved in deep sea shipping, where voyage travel times are long. Although voyage time windows can span several days, the long voyage travel times mean that we can expect few if any time feasible cycles involving voyages in the data instances. Similarly for maintenance requirements, we find that, although the time windows for these nodes are very wide, the required time for maintenance is long enough that time feasible cycles are unlikely. Therefore, we relax the subproblem to instead consider the regular shortest path problem with resource constraints (SPPRC), as this variant of the shortest path problem can be solved in pseudo polynomial time (see e.g. Desrochers and Soumis (1988); Irnich and Desaulniers (2005)). In Section 4.2.1 we discuss the possible existence of cycles and how to handle these. We solve the subproblems with a dynamic label setting algorithm on the underlying networks and refer the reader to Desaulniers et al. (1998), Irnich and Desaulniers (2005) and Irnich (2008) for a thorough introduction to the SPPRC, the related dynamic programming algorithms, and several associated concepts.

4.2.1 Subproblem networks

During construction of the subproblem networks, standard preprocessing techniques are applied to tighten the time windows. The VSR constraints are especially useful in this process since two consecutive voyages on a contract trade, r, must be separated in time by at least B_r and so, their individual time windows must reflect this. We describe this time window reduction technique in further details in Section 5.1.1.

The network node set for ship k includes the origin node, $o(k)$, and the destination, $d(k)$. For each contract or spot trade r that ship k is compatible with, the node set also includes a voyage node (r, i) for each voyage i that ship k is able to perform with respect to the voyage time window. Finally, if $\mathcal{N}_M^k \neq \emptyset$, then the node set also includes the maintenance node $(r, i) \in \mathcal{N}_M^k$.

For the origin node the time window is simply the open time for ship k, since there is no point in delaying departure. For each voyage and maintenance node, we calculate the earliest arrival at this node with ship k and in conjunction with the preprocessed time window for this voyage or maintenance, we can determine a ship specific time window for this voyage or maintenance node.

If $\mathcal{N}_M^k \neq \emptyset$, then ship k must undergo maintenance sometime during its schedule. Since there is no profit from visiting a maintenance node, we must force ship k to visit this node. Therefore, we introduce a binary maintenance resource that is equal to 1 once maintenance has been performed and 0 otherwise. This means, that the maintenance resource window at the destination node must be $[1, 1]$ while for the origin and maintenance node, it is $[0, 0]$. For voyage nodes, the maintenance resource window is $[0, 1]$.

Arcs are introduced to govern the transitions between the nodes. More specifically, the arc set contains all arcs that are time feasible for ship k and respect the internal order of voyages on the same trade. The latter means that, regardless of time feasibility, there can never be an arc from node $(r, i + 1)$ to node (r, i).

For two nodes, n_1 and n_2, connected by an arc in the network, this arc has a constant cost and time consumption and a well defined maintenance function and we denote these by $T_{n_1n_2}$, $C_{n_1n_2}$ and $M_{n_1n_2}$, respectively. If n_1 is the origin node, then n_2 is a voyage or maintenance node and the arc cost and time consumption correspond to sailing ballast from the port of the origin node to the first port associated with the voyage or maintenance node. If n_1 is a voyage node and n_2 is the destination node, then the cost corresponds to the negative of the profit from performing the voyage corresponding to n_1. The time consumption corresponds to the ship specific port time on this voyage plus the time to travel the voyage distance. If instead n_2 is another voyage or maintenance node, the cost must also include the additional cost of sailing ballast from the last port of the voyage corresponding to n_1 to the first port of the voyage or maintenance corresponding to n_2. Likewise, the time consumption must now include the time to travel from the last port of the voyage corresponding to n_1 to the first port of the voyage or maintenance corresponding to n_2. Finally, if n_1 is a maintenance node and n_2 is the destination node, then the cost is 0 while the time consumption is equal to the port time used during maintenance. If instead n_2 is a voyage node, then the cost and time consumption must, similar to before, also include the cost and time of traveling ballast from the maintenance port corresponding to n_1 to the first port of the voyage corresponding to n_2. Finally, $M_{n_1n_2}$ is equal to one on all arcs where n_1 is a maintenance node and equal to zero otherwise.

As already mentioned, we expect few if any time feasible cycles in our data instances. To detect the presence of potential cycles in a given subproblem network, a topological sort is performed on the node set. If a cycle is detected, the involved nodes are split into several duplicate ones, each with a smaller time window, until the cycle no longer exists; this process creates an acyclic network. It should be noted that with such a network we can generate schedules that visit multiple of these duplicate nodes, whereby the cycle in effect still exists. For maintenance nodes, there is however no profit. Therefore, we will never want to visit such nodes in the first place, and the maintenance resource ensures that we will visit exactly one of these. Should the shortest path solver generate schedules which visit several of the duplicate nodes for a voyage, these schedules will not be added to the master problem.

4.2.2 Dynamic Programming Algorithm

Given a dual solution to an optimized restricted master problem, the role of the subproblems is to identify whether or not a positive reduced cost schedule exists for any of the ships. This entails solving the SPPRC over the networks described above once the respective arc costs have been updated to reflect the dual solution. Updating the cost on arc (n_1, n_2) entails assigning it the negative of the reduced cost and we denote this cost as $\hat{C}_{n_1,n_2}$. Thereby,

$$\hat{C}_{o(k),(r,i)} = C^k_{o(k)ri} + \omega_k \qquad \forall\,(o(k),(r,i)) \in \mathcal{A}^k, \tag{29}$$

$$\hat{C}_{(r,i),(q,j)} = C^k_{riqj} - P^k_{ri} + \pi_{ri} \qquad \forall\,((r,i),(q,j)) \in \mathcal{A}^k, \tag{30}$$

$$\hat{C}_{(r,i),d(k)} = -P^k_{ri} + \pi_{ri} \qquad \forall\,((r,i),d(k)) \in \mathcal{A}^k. \tag{31}$$

As mentioned above, we solve the SPPRC using a dynamic programming algorithm. Such algorithms for this particular problem build new schedules for ship $k \in \mathcal{V}$ by starting with the trivial, partial schedule $s = \{o(k)\}$. Schedules are then built incrementally by extending partial schedules in all feasible ways. Partial schedules are represented by so-called *labels*. That is, for each partial schedule s_n ending in node n we associate a label $\mathcal{L}(s_n) = (\bar{C}(s_n), T(s_n), M(s_n))$. Here $\bar{C}(s_n)$ is the negative of the reduced cost for the schedule, i.e. the sum of the arc costs $\hat{C}_{n_1,n_2}$ for all $(n_1, n_2) \in s_n$. $T(s_n)$ and $M(s_n)$ denote, respectively, the arrival time at node n and the maintenance indicator on arrival at node n on schedule s_n.

Two partial schedules generated at the same node can be compared by defining a partial order relation between the respective labels. This partial order allows us to determine if one label dominates another that can, hence, be discarded. This dominance concept ensures that only the best schedules, i.e. Pareto optimal, are kept during the iterative process of the algorithm as only they can contribute to the optimal schedule. It is important to note that here "optimal" refers to the LP relaxed master problem, *without* constraints (23). The success of the algorithm relies on an efficient domination procedure for the labels to eliminate non-useful partial schedules. In our case we note that a schedule s_n ending at node n dominates another schedule s'_n also ending at node n if and only if $\mathcal{L}(s_n) \neq \mathcal{L}(s'_n)$, $\bar{C}(s_n) \leq \bar{C}(s'_n)$, $T(s_n) \leq T(s'_n)$ (since there is no cost for waiting) and $M(s_n) \geq M(s'_n)$.

In order to augment a partial path, label extension is necessary. Label extension is associated with a particular arc in the underlying network and utilizes specific resource extension functions that dictate how each resource level will change when traversing the arc. For the case at hand, the reduced cost resource is extended using simply $\bar{C}(s_{n_2}) = \bar{C}(s_{n_1}) + \hat{C}_{n_1,n_2}$. The time resource is extended using $T(s_{n_2}) = \max\{a_{n_2}, T(s_{n_1}) + T_{n_1 n_2}\}$, where a_{n_2} is the start of the time window on node n_2. This extension is deemed feasible if $T(s_{n_1}) + T_{n_1 n_2} \leq b_{n_2}$, where b_{n_2} is the end of the time window on node n_2. The maintenance resource is updated using $M(s_{n_2}) = M(s_{n_1}) + M_{n_1 n_2}$ which is deemed feasible if $M(s_{n_2}) \in [M^{min}_{n_2}, M^{max}_{n_2}]$, where $M^{min}_{n_2}$ and $M^{max}_{n_2}$ are, respectively, the lower and upper limit of the maintenance window for node n_2.

The dynamic programming algorithm we implement is hence a standard label setting algorithm, which begins at $o(k)$ with an initial label. Nodes are considered in topological order, and processed in turn. In processing a node, all non-dominated labels for the current node are extended, using the resource extension functions defined above and consider the node's set of outgoing arcs. When the algorithm terminates, several resource feasible and Pareto optimal schedules might exist differing in both reduced cost and time. In Section 4.3 we discuss what to do with these schedules. See Algorithm 1 for a general overview of our label setting algorithm.

4.3 Pricing Strategy

As already discussed, because of the VSR constraints we must consider any feasible schedule for a ship a valuable contribution to the master problem. This suggests that the shortest path solvers in the subproblems should return all resource feasible and Pareto optimal schedules with positive reduced costs, i.e. $\bar{C}(s) < 0$, to the RMP rather than just the best one or best ones. Preliminary tests verify this assumption and so, in each iteration we allow the subproblems to convert all resource feasible and Pareto optimal schedules with positive reduced costs to master problem columns.

Algorithm 1: Label Setting Algorithm

Input: Directed, Acyclic Graph $G = (\mathcal{N}, \mathcal{A})$, two nodes $o, d \in \mathcal{N}$
Output: Set of Pareto Optimal Schedules $\mathcal{S}$

```
Sorted Node List N̂ ← topologicalSort(G);
CreateInitialLabel(o);
for u ∈ N̂ and u ≠ d do
    L_u ← getLabels(u);
    for l ∈ L_u do
        if l is not dominated then
            for a ∈ outgoingArcs(u) do
                if extension(l, a) is feasible then
                    v ← headNode(a);
                    createLabel(l, a, v);
                    L_v ← getLabels(v);
                    dominanceCheck(L_v);
                end
            end
        end
    end
end
L_d ← getLabels(d);
S ← constructSchedules(L_d);
return S;
```

Preliminary tests also indicate that for this problem it is most efficient to solve all subproblems in each iteration, i.e. using the same dual values, as opposed to solving one subproblem in each iteration and then solving the master problem to obtain new dual values before solving the next subproblem. This makes sense since we can expect the master problem to grow rather quickly and therefore become quite time consuming to solve. Hence, it will be inefficient to solve it repeatedly just to obtain slightly better dual values from the columns generated by one single subproblem.

So, our pricing strategy is to solve all subproblems in each iteration and for each of these, we return all columns with positive reduced costs.

5 Branching

If the optimal solution to the restricted master problem is both integer (λ does not have to be integer as long as all positive λ's for each ship correspond to the same geographical route) and fulfills all the VSR constraints (23) and time window restrictions for spot vessels (24), the solution is also optimal for the full master problem (19)-(26) and thereby also for the original problem (1)-(18). However, if this is not the case, we must apply a branching scheme to restore feasibility. The VSR constraints and spot vessel time window constraints fit naturally into a time window branching scheme as presented by Gélinas et al. (1995). Furthermore, Gélinas et al. (1995) show that this branching procedure can also help enforce integrality, though only to a certain point. Therefore, we will apply time window branching and complement this with constraint branching (see Ryan and Foster (1981)) which is an effective branching strategy for restoring integrality on problems with similar structure to that of model (19)-(26).

5.1 Time Window Branching

The overall idea in time window branching is to split a given time window into two smaller time windows that each correspond to a new problem, i.e. to a new branch in the branch-and-bound tree. The trick is to select the time window and the split time in such a way that the current solution becomes infeasible in each of the two new problems, i.e. in a way that makes at least one

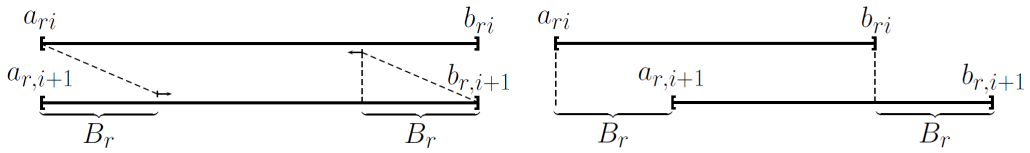
chosen schedule infeasible in each branch. Gélinas et al. (1995) note that their method works best on problems with small time windows and few cycles in the linear relaxation solution. As already mentioned, we expect few if any cycles in our data instances. However, time windows are relatively wide and so, it remains to be investigated if time window branching can work well for our problem. The method described by Gélinas et al. (1995) was developed to restore integrality and does not factor in VSR constraints. Therefore, we extend their method to incorporate such constraints just as e.g. Dohn et al. (2011) and Rasmussen et al. (2012) have done it to accommodate temporal dependencies for, respectively, vehicle routing and home care crew scheduling. Furthermore, we extend their methods to also account for the spot vessel time window constraints (24) and use a slightly modified approach that will improve efficiency of the branching scheme. Note that without the VSR constraints (23), the spot vessel time window constraints can never give rise to infeasibility. Therefore, the spot vessel time windows are only relevant when we consider violations of VSR constraints. Time window branching is not a complete branching strategy; i.e. fractionality can remain despite the fact that there are no time windows to branch on. That is why we complement this strategy with constraint branching.

5.1.1 Time Window Reduction

In order for the time window branching scheme to effectively restore feasibility with respect to the VSR constraints, we need to simultaneously apply a time window reduction rule based on these constraints. Table 1 therefore states the possible time window reductions for two consecutive nodes (r, i) and $(r, i+1)$ on trade r. The reduction process is illustrated in Figure 1.

Table 1: Time window reduction rule

	Node (r, i)	Node $(r, i+1)$
Old time window	$[a_{ri}, b_{ri}]$	$[a_{r,i+1}, b_{r,i+1}]$
New time window	$[a_{ri}, \min\{b_{ri}, b_{r,i+1} - B_r\}]$	$[\max\{a_{r,i+1}, a_{ri} + B_r\}, b_{r,i+1}]$



(a) Before reduction (b) After reduction

Figure 1: Time window reduction process for two consecutive nodes on a trade

We use the reduction rule not only for preprocessing but also in each branch-and-bound node where time window branching is applied. In fact, the time window branching scheme used for VSR violations can only work properly if we combine it with the reduction rule. In each new branch after time window branching on a node (r, i), we therefore apply the reduction rule not only to nodes $(r, i-1)$ and $(r, i+1)$ but iteratively to all nodes affected directly or indirectly by the time window changes for node (r, i) until no further reductions are possible. Note that if we did not require the VSR constraints to also hold for spot vessel voyages, then when a voyage (r, i) was assigned to a spot vessel, we would have to undo all previous time window reductions on trade r based on the time window of voyage (r, i)

5.1.2 Candidate time windows

Any node with a time window that can be split in a way that renders at least one currently chosen schedule infeasible in each of the two new branches, is a candidate for branching. If the current

master problem solution is fractional, then to fulfill constraints (20) or (21) there must be a node i (omitting the trade index r for now) that is visited by more than one schedule, one of them possibly a spot vessel schedule, or several times in a cycle by the same schedule. Hence, we must split the time window at this node so that there is only a single visit to the node. For each fractional schedule that visits node i, it might be possible to change the start time at node i slightly without rendering the schedule infeasible. For each visit to node i, we determine a *feasibility interval* defined by the earliest and latest possible start time at this node that will allow the corresponding schedule to remain feasible. For spot vessel schedules the feasibility interval simply corresponds to the (possibly reduced) time window at the node. The label setting algorithm used to generate schedules for fleet vessels, schedules each node visit as early as possible. This means that the feasibility interval at each node for regular schedules will simply correspond to allowing the ship to wait at long as possible. Assume now that node i is visited twice by two different schedules or twice by the same schedule. It does not matter if the two visits correspond to the same ship or to different ships, and so we can omit the k index here. Therefore, we abuse notation slightly by letting T_i^1 and T_i^2 denote the visit time for these two visits respectively. Now let $[T_i^1, u_1]$ and $[T_i^2, u_2]$ be the corresponding feasibility intervals and let $\epsilon > 0$ be a very small tolerance. If these two intervals are disjoint as shown in Figure 2, we can choose a split time for node i in $(u_1, T_i^2]$, say t_s, and create one branch where the time window for node i is $[a_{ri}, t_s - \epsilon]$ and the second visit is infeasible, and one branch where the time window is $[t_s, b_{ri}]$ where the first visit is infeasible. We can generalise this to state that any node for which two visits to the node have disjoint feasibility intervals, is a candidate for branching. A formal description and a proof for this, can be found in Gélinas et al. (1995). Note that split times within one of the feasibility intervals would also render one visit infeasible in each branch. However, during the next iteration of schedule generation, the visit, for which we selected a split time within its feasibility interval, could be regenerated though only a little later in time. This means that in one branch we will actually regenerate a fractional solution similar to the one from the parent node. Hence, it will be ineffective to use time window branching when the feasibility intervals are not disjoint. Note that in both Dohn et al. (2011) and Rasmussen et al. (2012) feasibility intervals are ignored whereby their branching scheme potentially becomes ineffective, both due to regeneration of fractionality but also because this approach causes them to consider more nodes as candidates for branching than if they had included the feasibility intervals. Since each candidate node must be further investigated to determine both the best split time for each node and in turn select the best node, this makes their approach more time consuming.

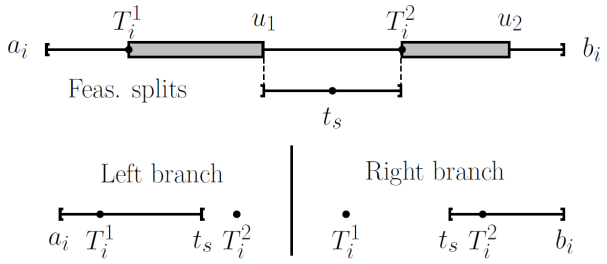


Figure 2: Time window branching due to fractionality

As already mentioned, the feasibility interval for a spot vessel visit will always correspond to the entire time window. Hence, such visits can never lead to disjoint feasibility intervals and so, time window branching will be ineffective. Therefore, when fractionality occurs partly due to a spot vessel visit, we will instead resort to constraint branching to restore feasibility.

When branching, we check if node i is on a trade r where VSRs exist. If it is, we can use the new time windows at node i to reduce the time windows of other nodes on trade r. Afterwards, all previously generated schedules violating these new time windows, are removed from the master problem in each new branch and the corresponding subproblems and spot vessel time windows are updated to reflect the new time windows.

Extending this concept to include VSR constraints and spot vessel time windows, further candidate time windows arise. Assume now that the VSR constraint for two consecutive voyages, i

and $i+1$, on trade r is violated and that $y_{ri} = y_{r,i+1} = 0$, i.e. no spot vessels are involved. If the current solution is integral, there is only one visit to each of these two nodes and these two visits per assumption violate the VSR constraint. If the current solution is fractional, there can be multiple visits to each of nodes (r,i) and $(r,i+1)$. For the VSR constraint to be violated, there must however be at least one pair of visits that on their own violate the VSR constraint. Whether the solution is integral or fractional, this means that there must exist positive $\lambda_{s_1}^{k_1}$ and $\lambda_{s_2}^{k_2}$ in the current RMP solution such that $T_{ris_1}^{k_1} + B_r > T_{r,i+1,s_2}^{k_2}$. Note that it is possible that $s_1 = s_2$ and $k_1 = k_2$. For the branching we generally do not need to know the specific schedule or ship index but simply that two visits for consecutive voyages on the same trade are violating the VSR. Therefore, we continue to abuse notation by letting T_i and T_{i+1} denote the visit times at these two nodes. To restore feasibility of the VSR constraint, we can force the start time of node (r,i) to be scheduled earlier, namely no later than $T_{i+1} - B_r$, we can postpone the start time of node $(r,i+1)$ so that it occurs at the earliest at $T_i + B_r$ or we can use a combination of these two time window alterations. In essence, the above corresponds to splitting the time window for node (r,i) and then reducing the time window of node $(r,i+1)$ accordingly so that the minimum time spread, B_r, is adhered to. Therefore, we use a split time, t_s, in the interval $[T_{i+1} - B_r + \epsilon, T_i]$ and create a left branch where the time window of node (r,i) is restricted to $[a_{ri}, t_s - \epsilon]$ whereby the schedule visiting node i becomes infeasible, and a right branch where the time window for node i is restricted to $[t_s, b_{ri}]$. In the left branch the time window for node $(r,i+1)$ remains unchanged while in the right branch we use the reduction rule from Section 5.1.1 to reduce it to $[t_s + B_r, b_{ri+1}]$ whereby the schedule visiting node $i+1$ becomes infeasible. Figure 3 demonstrates this process. Note that the process can be reversed to similarly enable a split of the time window at node (r,i) if instead the VSR violation stems from the node pair $(r,i-1)$ and (r,i).

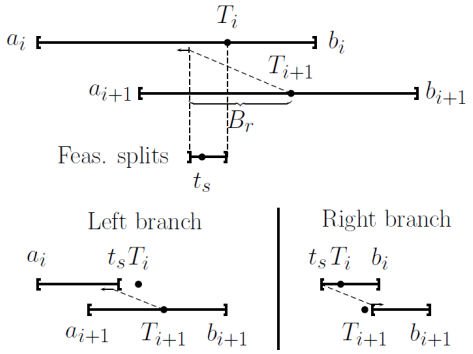


Figure 3: Time window branching due to VSR violation

In Dohn et al. (2011) they consider any node i a candidate node for branching if they can find s_1, s_2, k_1, k_2 with $\lambda_{s_1}^{k_1} > 0$ and $\lambda_{s_2}^{k_2} > 0$ such that (using our notation) $T_{r,i-1,s_1}^{k_1} + B_r > T_{ris_2}^{k_2}$. In Rasmussen et al. (2012) they reverse the search to consider any node i a candidate for branching if they can find s_1, s_2, k_1, k_2 with $\lambda_{s_1}^{k_1} > 0$ and $\lambda_{s_2}^{k_2} > 0$ such that (again using our notation) $T_{ris_1}^{k_1} + B_r > T_{r,i+1,s_2}^{k_2}$. However, it is important to understand that finding such a pair of visit times that on their own violate the corresponding VSR constraint, does not mean that the full VSR constraint is also violated, since this includes the weighted sum of all visit times currently in the solution. Obviously, if the current solution is integral, finding a pair of visit times that violate the VSR constraint, means that the full VSR constraint is also violated. However, when we factor in fractionality, this is not necessarily true. Therefore, the approach from both Dohn et al. (2011) and Rasmussen et al. (2012) means that the candidate list of nodes to branch on also includes nodes involved in VSR constraints that are not actually violated by the current solution. Two direct implications of this strategy are:

1. Branching on non-violated VSR constraints could lead to an ineffective branching scheme
2. Also considering nodes involved in non-violated VSR constraints to be branching candidates, could give rise to excessively long candidate lists. The selection of the best node for branching

involves further investigation of each of the node candidates to determine the best split time as well as the flow elimination from this best split time. Therefore, having a long candidate list will be time consuming.

Due to the above arguments, we refrain from using this process of running through all nodes and searching for pairs of visit times that on their own violate the corresponding VSR constraints. Instead, we will run through the VSR constraints and from each violated constraint for a node pair $((r,i),(r,i+1))$, we will consider both node (r,i) and $(r,i+1)$ a candidate for branching.

Now we extend further to include spot vessel schedules in the current (possibly fractional) solution. Therefore, we now assume that $y_{ri} > 0$ while $y_{r,i+1} = 0$ as previously and again consider the VSR constraint for (r,i) and $(r,i+1)$. To know whether or not this constraint is violated, we need to check whether or not a solution exists to the following small linear program (LP). To ease notation we have omitted the geographical route concept and simply sum over all schedules for each ship:

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i+1,s}\lambda^k_s, \tag{32}$$

$$a_{ri} \leq t^S_{ri} \;\leq\; b_{ri}. \tag{33}$$

We might find that $t^S_{ri} = a_{ri}$ is a feasible solution and therefore conclude that the VSR constraint is not violated. However, assume now that $y_{r,i-1} = 0$ and consider the similar LP for voyage pair $((r,i-1),(r,i))$:

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-1,s}\lambda^k_s + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri}, \tag{34}$$

$$a_{ri} \leq t^S_{ri} \;\leq\; b_{ri}. \tag{35}$$

If we insert $t^S_{ri} = a_{ri}$ in (34), we might find that the constraint is violated and thereby conclude that the VSR constraint for voyage pair $((r,i-1),(r,i))$ is violated even though the constraint need not be if we just select a different value for t^S_{ri}. Therefore, these two VSR constraints must be considered simultaneously. If $y_{r,i-1}$ is also positive, we must also include the VSR constraint for voyage pair $((r,i-2),(r,i-1))$ along with the spot vessel time window for $t^S_{r,i-1}$ and so it continues until we reach a voyage $(r,i-m)$ for which $y_{r,i-m} = 0$, or until we reach the first voyage on this particular trade. All of these VSR constraints and their corresponding spot vessel time windows form an LP for this particular *VSR group*:

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m,s}\lambda^k_s + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m+1,s}\lambda^k_s + t^S_{r,i-m+1}y_{r,i-m+1}, \tag{36}$$

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m+1,s}\lambda^k_s + t^S_{r,i-m+1}y_{r,i-m+1} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m+2,s}\lambda^k_s + t^S_{r,i-m+2}y_{r,i-m+2}, \tag{37}$$

$$\vdots \quad \vdots \quad \vdots$$

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-1,s}\lambda^k_s + t^S_{r,i-1}y_{r,i-1} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri}, \tag{38}$$

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i+1,s}\lambda^k_s, \tag{39}$$

$$a_{r,i-m+1} \leq t^S_{r,i-m+1} \;\leq\; b_{r,i-m+1}, \tag{40}$$

$$a_{r,i-m+2} \leq t^S_{r,i-m+2} \;\leq\; b_{r,i-m+2}, \tag{41}$$

$$\vdots \quad \vdots \quad \vdots \tag{42}$$

$$a_{r,i-1} \leq t^S_{r,i-1} \;\leq\; b_{r,i-1}, \tag{43}$$

$$a_{ri} \leq t^S_{ri} \;\leq\; b_{ri}. \tag{44}$$

If a feasible solution exists to the LP (36)-(44), the VSR group is not violated. If on the other hand no solution exists, at least one of the following statements must be true:

$$\exists g \in \{0,1,\ldots,m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + B_r > T^{k_2}_{r,i-g+1,s_2}, \quad (45)$$

$$\exists g \in \{1,2,\ldots,m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + 2B_r > T^{k_2}_{r,i-g+2,s_2}, \quad (46)$$

$$\exists g \in \{2,3,\ldots,m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + 3B_r > T^{k_2}_{r,i-g+3,s_2}, \quad (47)$$

$$\vdots \quad \vdots \quad \vdots$$

$$\exists g \in \{m-1,m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + mB_r > T^{k_2}_{r,i-g+m,s_2}, \quad (48)$$

$$\exists s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-m,s_1} + (m+1)B_r > T^{k_2}_{r,i+1,s_2}. \quad (49)$$

To exemplify, assume that the current solution has $y_{r,i-1}$ and y_{ri} positive while $y_{r,i-2} = y_{r,i+1} = 0$. Thereby, we have a VSR group consisting of the constraints for voyage pairs $((r,i-2),(r,i-1))$, $((r,i-1),(r,i))$ and $((r,i),(r,i+1))$. If the LP for this VSR group does not have a solution, there must exist s_1, s_2, k_1, k_2 with $\lambda^{k_1}_{s_1} > 0$ and $\lambda^{k^2}_{s_2} > 0$ such that at least one of the following is true:

$$T^{k_1}_{r,i-2,s_1} + B_r > T^{k_2}_{r,i-1,s_2} \quad (50)$$

$$T^{k_1}_{r,i-1,s_1} + B_r > T^{k_2}_{ris_2} \quad (51)$$

$$T^{k_1}_{ris_1} + B_r > T^{k_2}_{r,i+1,s_2} \quad (52)$$

$$T^{k_1}_{r,i-2,s_1} + 2B_r > T^{k_2}_{ris_2} \quad (53)$$

$$T^{k_1}_{r,i-1,s_1} + 2B_r > T^{k_2}_{r,i+1,s_2} \quad (54)$$

$$T^{k_1}_{r,i-2,s_1} + 3B_r > T^{k_2}_{r,i+1,s_2} \quad (55)$$

Constraint (52) corresponds to the illustration in Figure 3 and this figure must now be extended to include the time windows and visit times corresponding to constraints (51) and (53). The remaining violations will be handled when branching on nodes $(r,i-2)$, $(r,i-1)$ and $(r,i+1)$.

Spot vessels act as a shipping specific version of uncovered tasks, and these are not considered in Dohn et al. (2011), and in Rasmussen et al. (2012) they do not require uncovered tasks to be within the time windows. With our inclusion of spot vessels and time window restrictions it would be extremely time consuming to use the approach of Dohn et al. (2011) and Rasmussen et al. (2012) to run through all nodes and search for individual visit times that violate the constraints corresponding to the VSR group, i.e. constraints (45)-(49). Instead we run through each VSR group and only for the violated ones, we search for visit times that fulfill constraints (45)-(49), starting from (45). If we find one or several pairs of visit times that fulfill constraints (45), we have one or several pairs of candidate nodes for branching. In that case we do not check constraints (46)-(49) since these can be implicitly handled by branching to fulfill constraints (45) and using the time window reduction rule. However, if we do not find a pair of visit times that fulfill constraints (45), we move on to check constraints (46) and so the process continues until we find branching candidates.

As can be understood from the above, there can be numerous candidate time windows for branching in each iteration and we must decide on a selection strategy. However, the specific split time chosen within each candidate time window can be used in the selection procedure and so we postpone the discussion on how to select which time window to branch on, until after we have described how to choose the exact split time. For now, we therefore assume that we have chosen to split the time window for voyage i (ignoring again the trade index r).

5.1.3 Selecting the best split time within a time window

Assume for now that the current solution is fractional but fulfills all VSR constraints and spot vessel time restrictions. In this case, the window branching scheme simply aims at restoring integrality. A formal description of the split time selection procedure for this situation can be found in Gélinas et al. (1995); accordingly, we here give a more informal description.

Earlier we stated that any node, for which at least two non-spot vessel visits to the node have disjoint feasibility intervals, is a candidate for branching. When deciding on the best split time for

a given node, we consider the feasibility intervals of all visits to the node, again excluding visits from spot vessels. We illustrate the process in Figure 4 for a situation where node i is visited by four schedules included in the current solution with fractional values. We denote the four visit times at node i, respectively, T_i^1, T_i^2, T_i^3 and T_i^4. In Figure 4 the grey boxes represent the feasibility intervals for each of the four fractional visits to node i.

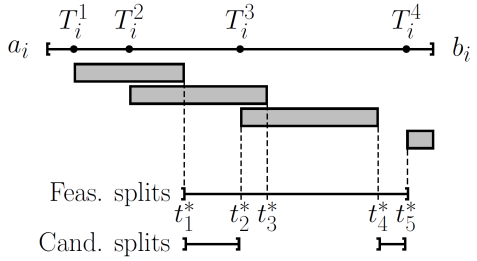


Figure 4: Choosing a split time to restore integrality

In order to render the current solution infeasible in both new branches, at least one of the four schedules must become infeasible in each of these branches. Therefore, the split time must be chosen somewhere in the interval between two disjoint feasibility intervals; hence, as depicted by the line marked 'Feas. splits' in Figure 4. Note that split times in this interval lie outside the feasibility interval corresponding to visit time T_i^1. Thereby, we can never regenerate a similar visit in the right branch, where visit time T_i^1 becomes infeasible. Comparing the intervals $(t_2^*, t_3^*]$, $(t_3^*, t_4^*]$ and $(t_4^*, t_5^*]$ we see that split times within these three intervals have the same affect with respect to immediate flow elimination. However, if we select a split time in $(t_2^*, t_3^*]$ we risk regenerating the schedules corresponding to both T_i^2 and T_i^3 by redistributing waiting time. If instead we select a split time in $(t_3^*, t_4^*]$, we cannot regenerate the schedule corresponding to T_i^2, and selecting a split time in $(t_4^*, t_5^*]$ we cannot regenerate either of the schedules corresponding to T_i^2 and T_i^3. In order to avoid regenerating schedules that lead to fractionality, we will therefore never consider selecting a split time within $(t_2^*, t_4^*]$. More generally, the open ended candidate intervals for split time should not include starting or end points of feasibility intervals and we should seek split times on the boundaries of the feasibility intervals. Thereby, we generally seek split times that lie outside the feasibility intervals. Hence, we can reduce the set of candidate split times to the two smaller intervals shown in Figure 4 as 'Cand. splits'. Within each of these two intervals the amount of flow eliminated is independent of the chosen split time. However, as discussed, the label setting algorithm used to generate the schedules, ensures that each visit is scheduled as early as possible. Thereby, no visit can be regenerated at an earlier point in time but it can however be postponed. Selecting the split time as late as possible within a given candidate interval therefore generally minimises the risk that an eliminated visit will be regenerated. In Figure 4 this means that in the interval $(t_1^*, t_2^*]$ we select t_2^* as the split time while in $(t_4^*, t_5^*]$ we select t_5^*. Note that t_2^* and t_5^* both correspond to visit times. We generalise this to say that each currently selected visit time, T_i, after the feasibility interval of the earliest selected visit at the node, is a candidate split time. By using this approach, we only need to calculate the feasibility interval of the first visit to the node.

Now we extend to include VSR violations and assume that nodes $(r, i-1)$, (r, i) and $(r, i+1)$ together form a VSR group. Figure 5 therefore extends the example from Figure 4 to include VSR violations for node pair $(r, i-1)$ and (r, i) as well as for node pair (r, i) and $(r, i+1)$. Here T_{i-1}^1 and T_{i+1}^1 denote, respectively, the visit time of a visit to node $i-1$ and $i+1$ for two schedules included in the current solution. Note that for VSR violations we do not care about feasibility intervals since it is the exact visit time at the node that impacts the VSR constraint. Therefore, the feasibility intervals are not included in Figure 5. There are four VSR violations and each of these leads to an interval of feasible split times. These are marked 'Feasible split intervals' in the bottom of the figure. The start and end points of these four intervals are marked as t_6^* to t_{11}^* and together they define five intervals, i.e. $(t_6^*, t_7^*]$ to $(t_{10}^*, t_{11}^*]$, which each have a distinct elimination of flow. Using the same reasoning as above, within each of these five intervals we prefer to select the latest time. Therefore, in Figure 5, t_7^* to t_{11}^* are all candidates for split times. We note that

t_7^*, t_{10}^* and t_{11}^* all correspond to visit times at the node while t_8^* and t_9^* correspond to, respectively, $T_{i+1}^1 - B_r$ and $T_{i-1}^1 + B_r$. We can generalise this to say that, limiting the search to schedules included in the current solution with a positive value, the start time of every visit to the node, except the first, is a candidate split time. Furthermore, for any visit at node $i-1$, for which the visit time T_{i-1} causes a violation of the VSR for nodes $i-1$ and i, $T_{i-1}^1 + B_r$ is also a candidate split time. Similarly, any visit at node $i+1$, for which the visit time T_{i+1} causes a violation of the VSR for nodes i and $i+1$, $T_{i+1} - B_r$ is a candidate split time. Furthermore, for various integer values of m, depending on the size of the VSR group that node i belongs to, $T_{i\pm m} \pm mB_r$ can also be candidate split times. Note that this extends the candidate split times derived from fractionality since all start times at the node except the first one is now a candidate split time.

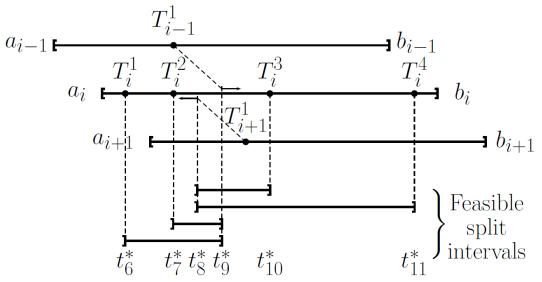


Figure 5: Choosing a split time to restore VSR

Assume now that the schedules corresponding to T_i^1, T_i^2, T_i^3 and T_i^4 are in the current solution with values 0.1, 0.1, 0.4 and 0.4, respectively. Furthermore, assume that the two visits at nodes $i-1$ and $i+1$ are the only visits to these nodes, i.e. that the schedules corresponding to these visits are both in the solution with a value of 1. Table 2 then shows the infeasible visits and the corresponding eliminated flow in, respectively, the left and right branch when choosing one of the candidate split times derived from Figure 5 (which also covers the candidate split times from Figure 4).

Note that since the branching applied to restore feasibility with respect to VSRs only factors in the exact visit times and ignores the feasibility intervals, this type of branching allows us to regenerate similar schedules with visit times just slightly postponed. This is sufficient for the VSR constraints, however it might not be sufficient to rule out regeneration of fractionality. As an example, consider the split time candidate $t_7^* = T_i^2$ where the right branch will exclude visit time T_i^1. According to Figure 4, T_i^2 is within the feasibility interval of the schedule, say s_1, corresponding to visit time T_i^1. Therefore, we can easily generate a new schedule similar to s_1 with the visit at node i postponed slightly. Thereby, the exact same fractionality will occur again. Therefore, when branching on a node where there is no VSR violations, we reinclude the feasibility interval aspect to more effectively eliminate fractionality.

Table 2: Flow elimination from candidate split times

	Infeasible visits		Eliminated flow		
Candidate	Left branch	Right branch	Left branch	Right branch	Minimum
t_7^*	$T_i^2, T_i^3, T_i^4, T_{i-1}^1$	T_i^1	1.9	0.1	0.1
t_8^*	T_i^3, T_i^4, T_{i-1}^1	T_i^1, T_i^2	1.8	0.2	0.2
t_9^*	T_i^3, T_i^4, T_{i-1}^1	T_i^1, T_i^2, T_{i+1}^1	1.8	1.2	**1.2**
t_{10}^*	T_i^3, T_i^4	T_i^1, T_i^2, T_{i+1}^1	0.8	1.2	0.8
t_{11}^*	T_i^4	$T_i^1, T_i^2, T_i^3, T_{i+1}^1$	0.4	1.6	0.4

To motivate flow elimination while maintaining a balanced search tree, we prefer the candidate that eliminates the most flow in the worst of the branches, i.e. the candidate with the highest value of flow elimination in the branch where it eliminates the least flow. This number is given in Table 2 in column 'Minimum' for each candidate, and we see that the best worst case flow elimination is achieved for t_9^*.

Note that Table 2 only lists flow immediately eliminated and not flow implicitly eliminated from further time window reductions. Since each trade can consist of many voyages, calculating the full flow elimination for each candidate split time for each candidate node, can be time consuming. Therefore, we refrain from such extensive calculations and simply consider the direct flow elimination. This is another reason to use the best worst case flow elimination as a selection criteria, since we know that regardless of the implicit flow elimination, we can never do worse than this.

5.1.4 Selecting the best time window for branching

Now that we know how to find candidate time windows for branching and also how to actually split the chosen time window, we are ready to choose which time window to branch on. Again, aiming at eliminating as much flow as possible while maintaining a well balanced search tree, we select the time window that has the best worse case flow elimination. I.e. the approach from Table 2 for selecting the best split time for a given time window, is also used to select the best time window to split.

5.2 Constraint Branching

If slack variables, y_{ri}, are inserted into constraints (21), the RMP is modelled as a set partitioning problem with generalised upper bound constraints (22). Due to the generalised upper bound constraints (22), the submatrix for each ship is perfect. Thereby, fractional solutions can only appear across submatrices for different ships and never within one of the individual ship submatrices. This means that the LP solution can only be fractional if two or more ships are competing for the same voyage. Note the word 'voyage' since ships can never compete for the individual maintenance requirements. We refer the reader to Padberg (1973) and Conforti et al. (2001) for a discussion on perfect matrices and their properties. This strong integer property of the RMP constraint matrix means that the upper bounds in the branch-and-bound algorithm are very tight and that we can expect to reach integral solutions after only a few iterations of branching. We exploit this underlying structure of the constraint matrix in the branching scheme to apply so called constraint branching, see Ryan and Foster (1981). Note that this strong integer property also means that we can refrain from checking for 'equal geographical routes' for each ship.

5.2.1 Candidate voyage-ship pairs

If the current solution is fractional, there must be a voyage $(r, i) \in \mathcal{N}_C \cup \mathcal{N}_O$ that is performed by several ships. Since, by definition, each spot vessel (we can also view slack variables as a form of spot vessel schedules) can only perform one voyage, the spot vessels cannot compete with each other for voyages. Therefore, at least one of the ships currently competing for voyage (r, i) must be a non-spot vessel and we denote this ship k. In an integral solution, the voyage can only be performed by one ship and therefore ship k can either perform or *not* perform voyage (r, i). For each ship $k \in \mathcal{V}$ and voyage $(r, i) \in \mathcal{N}_C \cup \mathcal{N}_O$ we introduce the sum

$$S_{ri}^k = \sum_{s \in \mathcal{S}^{k\prime}} A_{ris}^k \lambda_s^k.$$

If the current solution is fractional, there must exist a voyage (r, i) and a ship k for which $0 < S_{ri}^k < 1$. The branching strategy is then to construct a left branch where ship k is forced to perform voyage (r, i) and a right branch where ship k is not allowed to perform voyage (r, i). In the left branch this means that $A_{ris} = 1$ for all $s \in \mathcal{S}^k$. Thereby, we can remove all schedules for k that do not include voyage (r, i) and all columns for other ships that do include voyage (r, i). This also means removing the spot vessel schedule for voyage (r, i) or equivalently setting $y_{ri} = 0$. Furthermore, we remove voyage (r, i) in all subproblems networks not corresponding to ship k.

We rely on the dual variables to eventually enforce the construction of schedules for ship k that include voyage (r, i). In the right branch, we instead have $A_{ris} = 0$ for all $s \in \mathcal{S}^k$. This is the opposite process of the left branch, and so we instead remove all schedules for k that *do* include voyage (r, i). Furthermore, we remove voyage (r, i) in the subproblem network corresponding to ship k while we cannot make any changes in the networks corresponding to other ships.

To ease notation, we let S^S_{ri} denote the similar sum for spot vessels (including slack variables) even though this simply correspond to y_{ri}. When $0 < S^S_{ri} < 1$, we can then use the same branching approach as just described for regular ships though the corresponding updates of the RMP and subproblems must of course be modified accordingly. Also note that since spot vessels cannot compete for the same voyages, we can never have fractional y-variables without also having at least one fractional λ-variable. Therefore, in a fractional solution we can always find at least one voyage (r, i) and one regular ship $k \in \mathcal{V}$ for which $0 < S^k_{ri} < 1$.

5.2.2 Selecting the voyage-ship pair for branching

Competition for a voyage (r, i) requires at least two different ships. Thereby, in a fractional solution there must be at least two distinct ships (one of them possibly a spot vessel), k_1 and k_2, for which $S^{k_1}_{ri}$ and $S^{k_2}_{ri}$ are fractional. Therefore, in any fractional solution we have at least two candidate voyage-ship pairs for branching and we must select one of these.

If S^k_{ri} is close to 1, the solution favours ship k for performing voyage (r, i); hence, forcing ship k to perform the voyage, as we do in the left branch, will probably not change the solution much and thereby not the upper bound either. In the right branch, where we force ship k not to perform the voyage, we can on the other hand expect to see a greater impact on the upper bound. This could potentially create an unbalanced search tree. The situation is similar, though reversed, if S^k_{ri} is close to 0. To maintain a balanced search tree we therefore prefer branching on candidates that are as fractional as possible, i.e. as close to 1/2. We denote the list of candidate pairs by $\mathcal{C}$, i.e

$$\mathcal{C} = \{((r, i), k) \in \mathcal{N}_C \cup \mathcal{N}_O \times \mathcal{V} \cup \{S\} : 0 < S^k_{ri} < 1\}.$$

We then select the candidate pair that leads to the most balanced search tree, i.e. select

$$((r, i), k)^* = \arg \min_{((r,i),k)\in\mathcal{C}} \left\{ \left| S^k_{ri} - \frac{1}{2} \right| \right\}.$$

5.3 Branching Strategy

We use depth-first search and consider three different branching strategies with different priorities to the different branching schemes:

1. Constraint branching first, time window branching second.
2. Time window branching first, constraint branching second.
3. VSR related time window branching first, constraint branching second.

The first strategy initially uses constraint branching to eliminate fractionality and then turns to time window branching to handle violations of VSR constraints. Note that in this case, the time window branching scheme will be simplified since only one visit to each node exists and we do not have to consider feasibility intervals. The second strategy uses time window branching to handle both VSR violations and fractionality and only turns to constraint branching if we run out of time windows to branch on. The third strategy starts by handling all violations of VSR constraints while ignoring fractionality, i.e. disregarding feasibility intervals and in general nodes that are not involved in VSR violations. Afterwards, all fractionality is eliminated through constraint branching. Note though that the VSR related time window branching will simultaneously help to eliminate fractionality.

Preliminary tests show that the strategy that first performs VSR related time window branching and then constraint branching, consistently outperforms the other strategies, and so we use this strategy for our computational study.

6 Computational Study

In this section, we describe data and results from our computational study to evaluate the performance of the developed algorithm. As a reference point for this evaluation, we compare our method with the a priori path generation method from Norstad et al. (2013).

6.1 Data instances

To properly evaluate the performance of the devised algorithm, we test it on 16 problem instances of varying complexity and size. These instances have been generated by the test instance generator described in Norstad et al. (2013) which is based on industry data. Table 3 presents the main characteristics of the 16 data instances. The column labels are almost self explanatory but for completeness sake we note that from left to right they give the instance number, the number of ships, the number of trades, the number of voyages where the number in parenthesis gives the number of spot voyages and finally, the length of the planning horizon in days. Note that this horizon is defined as the length of the period that contains the earliest allowed starting time for each voyage. Thereby, planning will continue well beyond this horizon since voyages can be performed later than the earliest allowed time and must also be completed.

Table 3: Data instance characteristics

No.	Ships	Trades	Voyages	Horizon
1	10	4	21(5)	90
2	10	7	32(9)	90
3	10	4	19(0)	90
4	10	4	25(0)	120
5	10	5	34(9)	120
6	10	5	36(5)	120
7	10	5	42(11)	150
8	10	6	52(13)	150
9	10	6	47(8)	150
10	25	8	49(10)	90
11	25	8	44(11)	90
12	25	8	53(11)	90
13	25	8	57(10)	105
14	25	8	55(5)	105
15	25	8	64(12)	120
16	32	13	55(12)	90

6.2 Computational Results

In order to evaluate the performance of the devised algorithm, we run it on all 16 data instances, and compare the results obtained from these tests with results from using the path generation method from Norstad et al. (2013).

6.2.1 Results from devised Branch-and-Price method

The results from our developed Branch-and-Price method are obtained using a PC with 4.0 GB RAM and an Intel(R) Core(TM)2 Duo CPU P8600, 2.4 GHz processor under a 64 bit Windows 7. The algorithm is implemented in C++ using Cplex 12.4 with default settings to solve the master problem and the time window branching parameter ϵ set to 0.001, which corresponds to 1.44 minutes on our data instances. Table 4 shows the results from running our algorithm on the 16 data instances. The column 'Inst' gives the instance number while columns 'Objective' and

'Gap' contains, respectively, the objective value of the best solution found and the integrality gap in percentage for this solution. In column 'Time' we list the time used to solve the problem. We allow a maximum running time of 3600 seconds and if the algorithm runs out of time before closing the integrality gap, we list the time it took to find the best solution and put a '*' to indicate that we ran out of time. Column 'Vars' lists the number of variables in the final master problem, i.e. the number of columns generated. Column 'Nodes' gives the number of explored Branch-and-Bound nodes and, finally, in columns 'Time_{LP}' and 'Time_{Sub}' we have the time spent solving, respectively, the master problem and the subproblems.

Table 4: Results from Branch-and-Price method

Inst	Objective	Gap	Time	Vars	Nodes	Time_{LP}	Time_{Sub}
1	12 423 322	-	0	377	153	0	0
2	17 769 109	-	0	401	33	0	0
3	12 993 990	-	0	231	9	0	0
4	15 817 224	-	1	1512	311	0	0
5	15 422 210	-	1	887	139	0	0
6	21 555 416	-	0	660	29	0	0
7	20 252 953	-	0	476	5	0	0
8	21 026 411	-	0	330	1	0	0
9	23 201 012	-	2	1415	275	0	1
10	36 703 643	0.79	0*	27821	75780	2243	283
11	28 988 536	-	0	780	4	0	0
12	38 871 251	-	7	2464	967	1	4
13	37 650 390	-	0	1393	3	0	0
14	37 505 930	0.23	0*	4510	175184	1088	931
15	43 260 992	0.07	1*	4326	167019	979	1096
16	40 909 308	0.23	1046*	73059	15441	2131	149

From Table 4 we first see that on four instances the algorithm is unable to close the integrality gap after 3600 seconds. However, all four integrality gaps are well below 1% and hence very small. Next, we see that, aside from instance 16, the algorithm very quickly finds a good if not optimal solution. On instance 16, the time to find the best solution is however 1046 seconds. Looking at the number of generated variables, we see that it ranges from as low as just 231 variables up to 73,059. The number of Branch-and-Bound nodes shows a similar diversity of the test instances as this number ranges from just 1 node, i.e. problem solved at root node, up to as many as 175,184 nodes. Combining the number of variables and the number of nodes with the time usage for master problem and subproblems, we note that the four occasions where the algorithm runs out of time can be divided into two different categories. For problem instances 10 and 16, so many variables are generated that the master problem becomes so large, that the computation time to solve it repeatedly dominates the procedure. For these instances, it would probably help to use a higher value for the time window branching parameter ϵ. This will in general speed up the algorithm; however, it can sacrifice optimality. On the other hand, for instances 14 and 15, the number of generated variables is not very high. Instead, the number of explored nodes is huge and suggests that for these instances it would help to implement a different search strategy for the branch-and-bound tree, e.g. Best-First-Search. It could also help to use different selection rules for time windows and split times in the time window branching scheme, or for ship-voyage pairs in the constraint branching scheme. More generally, the branching scheme might be improved by using strong branching (see Achterberg et al. (2005)). Overall we note that our algorithm is able to find very good if not optimal solutions extremely fast, though one instance requires longer time.

As already mentioned, Norstad et al. (2013) find that voyage separation requirements can significantly improve the spread of the voyages and at only marginal profit reductions. Although we do not wish to repeat their analysis here, we note that we arrive at similar findings after running the 16 instances again without voyage separation requirements. In fact, the profit reduction is below

1% on all 16 instances. The complexity added from the voyage separation requirements is however not insignificant and this is most notable for instance 10 where we ran out of time when voyage separation requirements were included in the problem. Running instance 10 again, though this time without these separation requirements, we are able to solve the instance to optimality using just 0.12 seconds and exploring just 7 nodes. On instance 16 we still run out of time but now the best solution is found after just 535 seconds, where it was 1046 seconds with the separation requirements included.

6.2.2 Results from A Priori Path Generation method

To properly evaluate the efficiency of our Branch-and-Price algorithm we want to compare our results with results obtained from using the A Priori Path Generation (APPG) method described in Norstad et al. (2013). This method first a priori generates feasible paths and then uses a commercial solver to solve the path flow formulation containing these generated paths. We use Section 6.2.3 to compare the results from the two methods while in this section we focus solely on the APPG method and the results obtained from it.

The authors from (Norstad et al., 2013) have provided us with results from running their APPG method on the same 16 data instances that we use here. Their results are obtained using a comparable DELL Latitude Laptop with Intel Core i5 CPU (4x2.40 GHz), 4GB DDR2 RM running on Windows 7. Their path generator is implemented in C# while the path flow model is solved with Xpress MP 7.0 64 bit. Results from running their method also with a time limit of 3600 seconds on the 16 data instances considered here are given in Table 5. The column headers are the same as in Table 4 though we have added a column denoted 'Constrs' which lists the number of constraints in the path flow model.

Table 5: Results from A Priori Path Generation method

Inst	Objective	Time	Vars	Constrs
1	12 423 322	0	946	4866
2	17 638 764	1	2897	10936
3	12 993 990	0	1150	4035
4	15 817 224	15	3139	6820
5	15 422 210	1	6050	12302
6	21 555 416	2	12285	13768
7	20 252 953	2	14344	18567
8	20 062 828	4	14781	28194
9	22 916 245	44	15554	23138
10	34 894 744	4	9461	62462
11	28 906 248	3	7175	50569
12	38 871 251	19	11291	72853
13	37 626 000	13	32346	84093
14	37 465 639	678	26344	78412
15	40 822 868	119	52625	105636
16	40 883 752	13	8314	103590

Ignoring the objective function values for now, we first note from the 'Time' column in Table 5 that the time usage from this method is generally a lot longer than from our algorithm. However, this method never runs out of time and therefore never exhibits any integrality gaps. It would therefore be natural to conclude that the APPG method is slower but more stable than our method. However, although Norstad et al. (2013) present their APPG method as an exact one, we claim that this is not the case. We discuss this further in the next section when comparing results from the APPG method and our Branch-and-Price method. For now we focus on the APPG method itself and further investigate the actual implementation of it. From Norstad et al. (2013) we have the following quote describing the details of their implementation:

"In order to generate the parameters for the path flow model a path generator program has been implemented. All feasible paths for each ship are generated a priori to the optimization. Since the required maintenance operations and also some the voyages can have quite wide time windows, there may be several feasible sequences or paths a ship can follow while performing the same set of voyages. A simple dominance test is therefore performed to make sure that only the most profitable one is passed on to the optimization model."

As discussed in Section 4, without the voyage separation requirements it is sufficient to include the profit maximising schedule for each ship and cargo set. However, with the separation requirements included in the problem this is no longer true since the actual timing of port calls in a schedule for one ship can affect the timing of port calls for schedules of other ships. Instead, any feasible schedule for a ship can potentially be part of the optimal solution and the master problem, or in their case the path flow formulation, can contain several schedules all corresponding to the same set of voyage and maintenance stops. Therefore, the 'simple dominance test' mentioned in the quote above actually sacrifices optimality of the APPG method as implemented by Norstad et al. (2013). With the voyage separation requirements included the a priori generation approach should be quite time consuming which is why we turned to dynamic column generation. However, because of this dominance test the APPG method becomes manageable even with these separation requirements included. So, this dominance test makes their APPG implementation extremely efficient, though it does sacrifice optimality.

6.2.3 Comparing the two methods

To properly evaluate our devised Branch-and-Price method we use this section to compare it with the APPG method from Norstad et al. (2013). Table 6 therefore summarises the key values from Tables 4 and 5 from running each of these two methods on the considered 16 data instances. For the APPG method we have added a column denoted 'Gap' which contains the integrality gap in percentage from comparing the APPG solution with the bound obtained from our Branch-and-Price method. Furthermore, the last column denoted 'Obj. Incr.' lists the percentage increase in objective function value from using our algorithm compared to the APPG algorithm.

Table 6: Comparing solution methods

	A Priori Path Generation				Branch-and-Price				
Inst	Obj	Gap	Time	Vars	Obj	Gap	Time	Vars	Obj. Incr.
1	12 423 322	-	0	946	12 423 322	-	0	377	-
2	17 638 764	0.7	1	2897	17 769 109	-	0	401	0.7
3	12 993 990	-	0	1150	12 993 990	-	0	231	-
4	15 817 224	-	15	3139	15 817 224	-	1	1512	-
5	15 422 210	-	1	6050	15 422 210	-	1	887	-
6	21 555 416	-	2	12285	21 555 416	-	0	660	-
7	20 252 953	-	2	14344	20 252 953	-	0	476	-
8	20 062 828	4.6	4	14781	21 026 411	-	0	330	4.8
9	22 916 245	1.2	44	15554	23 201 012	-	2	1415	1.2
10	34 894 744	5.7	4	9461	36 703 643	0.8	0*	27821	5.2
11	28 906 248	0.3	3	7175	28 988 536	-	0	780	0.3
12	38 871 251	-	19	11291	38 871 251	-	7	2464	-
13	37 626 000	0.1	13	32346	37 650 390	-	0	1393	0.1
14	37 465 639	0.3	678	26344	37 505 930	0.2	0*	4510	0.1
15	40 822 868	5.7	119	52625	43 260 992	0.1	1*	4326	6.0
16	40 883 752	0.3	13	8314	40 909 308	0.2	1046*	73059	0.1

From Table 6 we first note that, compared to the bound obtained from our algorithm, the APPG method now experiences integrality gaps for 9 out of the 16 instances. We also note that these gaps are consistently larger than our gaps which is consistent with the fact that our solutions are consistently equal to or better than the ones obtained from the APPG method. In fact, the profit increase from using our algorithm compared to the APPG method is as high as 6% for one instance. Focusing instead on the time usage of the two algorithms, we see that on the instances where our algorithm does not run out of time, the APPG method consistently uses the same or longer time. On the four instances where our algorithm runs out time, we manage to find a better solution than the APPG method and for three out of these four instances this better solution is found much faster than the solution from the APPG method. However, for instance 16 this is not true. Finally, we turn to the variable count for each algorithm and note that, aside from instances 10 and 16 where our algorithm runs out time due to an extensive generation of variables, the APPG method consistently includes much more variables than our method. This is consistent with our dynamic column generation approach which only generates columns as needed.

Overall we find that our devised Branch-and-Price algorithm provides very good if not optimal solutions extremely fast. In fact, it consistently finds equal or better solutions than the APPG method and for all but one instance the solution is found in the same or a shorter time than the APPG method uses.

7 Concluding Remarks

In this paper we have considered the tramp ship routing and scheduling problem with voyage separation requirements. These separation requirements enforce a minimum time spread between voyages on the same trade. This is done in an attempt to improve the situation for the charterer who faces increased inventory costs if voyages are not performed 'fairly evenly spread' in time. In this respect, the separation requirements correspond to a crude way of viewing the tramp ship routing and scheduling problem in the broader context of the supply chain.

We have developed a new and exact method for this problem. It is a Branch-and-Price procedure with a dynamic programming algorithm to dynamically generate columns and with the voyage separation requirements relaxed in the master problem and instead enforced through a modified time window branching scheme. Computational results from 16 data instances show that our algorithm finds very good if not optimal solutions extremely fast though one instance requires longer time. On four instances our algorithm is unable to prove optimality within 3600 seconds and, although the integrality gap is small for all four instances, it would still be interesting to explore different search strategies for the branch-and-bound procedure and different modifications to the branching schemes.

Running our algorithm on all 16 instances without the voyage separation requirements showed that the profit reduction from including the separation requirements is below 1% on all instances. This is consistent with the findings in Norstad et al. (2013) who conclude that voyage separation requirements can significantly improve the spread of the voyages and at only marginal profit reductions. There is however a significant increase in problem complexity from these additional requirements and for one instance this increase caused the algorithm to require more than 3600 seconds to solve a problem that was otherwise solvable within just 0.12 seconds without the additional requirements.

We compared our Branch-and-Price method to the APPG method from Norstad et al. (2013) that we have shown is not optimal. Results from comparing the two methods on the 16 data instances confirm this. In fact, our solutions are consistently equal to or better than the ones obtained from the APPG method and the profit increase from using our algorithm compared to the APPG method is as high as 6% for one instance. Furthermore, on all but one instance, our solution is found in equal or shorter time than what the APPG method uses.

Overall we have developed a new, exact method for the tramp ship routing and scheduling problem with voyage separation requirements. This method is extremely fast at finding very good if not optimal solutions, although one instance requires longer time. An interesting extension of this work would be to explore different branch-and-bound search procedures as well as different modifications to the branching schemes. Finally, it would also be interesting to investigate the

effect of modifying the APPG method to obtain optimal solutions, i.e. removing the dominance test discussed in Section 6.2.2.

Acknowledgements

The research presented in this paper has been partly funded by The Danish Maritime Fund and we gratefully acknowledge their financial support. We also want to acknowledge Professor Kjetil Fagerholt and Inge Norstad, both from the Norwegian University of Science and Technology (NTNU), for all their help on this project. They have provided advice, insight as well as data, and we are thankful for all the time they set aside for us, especially during our visits to NTNU.

References

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

K.K. Castillo-Villar, R.G. González-Ramírez, P.M. González, and N.R. Smith. A heuristic procedure for a ship routing and scheduling problem with variable speed and discretized time windows. *Mathematical Problems in Engineering*, 2014. doi: http://dx.doi.org/10.1155/2014/750232.

M.E. Cóccola, R. Dondo, and C.A. Méndez. A milp-based column generation strategy for managing large-scale maritime distribution problems. *Computers & Chemical Engineering*, 2014. doi: http://dx.doi.org/10.1016/j.compchemeng.2014.04.008.

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

M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Maritime transportation. In C. Barnhart and G. Laporte, editors, *Transport. Handbooks in Operations Research and Management Science, vol. 14*, chapter 4, pages 189–284. Elsevier, North-Holland, Amsterdam, 2007.

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

M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vušković. Perfect, ideal and balanced matrices. *European Journal of Operational Research*, 133(3):455–461, 2001.

J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon, and F. Soumis. Vrp with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 7, pages 157–194. Society for Industrial and Applied Mathematics, 2002.

G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, and D. Villeneuve. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57 – 94. Kluwer Academic Publishers, 1998.

G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors. *Column Generation*. Springer, New York, 2005.

M. Desrochers and F. Soumis. A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, 35:242 – 254, 1988.

A. Dohn, M.S. Rasmussen, and J. Larsen. The vehicle routing problem with time windows and temporal dependencies. *Networks*, 58(4):273–289, 2011.

M. Drexl. Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research*, 227(2):275–283, 2013.

M. Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978, 1994.

K. Fagerholt and D. Ronen. Bulk ship routing and scheduling: Solving practical problems may provide better results. *Maritime Policy and Management*, 40(1):48–64, 2013.

S. Gélinas, M. Desrochers, J. Desrosiers, and M.M. Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995.

S. Irnich. Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, chapter 2, pages 33 – 66. Springer, 2005.

K. Kang, W.-C. Zhang, L.-Y. Guo, and T. Ma. Research on ship routing and deployment mode for a bulk. pages 1832–1837, 2012. Int. Conf. Manage. Sci. Eng. - Annu. Conf. Proc., Dallas, TX.

I.K. Moon, Z.B. Qiu, and J.H. Wang. A combined tramp ship routing, fleet deployment, and network design problem. *Maritime Policy and Management*, 2014. doi: 10.1080/03088839.2013.865847.

I. Norstad, K. Fagerholt, L.M. Hvattum, H.S. Arnulf, and A. Bjørkli. Maritime fleet deployment with voyage separation requirements. *Flexible Services and Manufacturing Journal*, 2013. doi: 10.1007/s10696-013-9174-7.

M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1): 199–215, 1973.

M.S Rasmussen, T. Justesen, A. Dohn, and J. Larsen. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, 219:598–610, 2012.

L.B. Reinhardt, T. Clausen, and D. Pisinger. Synchronized dial-a-ride transportation of disabled passengers at airports. *European Journal of Operational Research*, 225(1):106–117, 2013.

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

D. Ronen. Ship scheduling: The last decade. *European Journal of Operational Research*, 71(3): 325–333, 1993.

D.M. Ryan and B. Foster. An integer programming approach to scheduling. *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling. Proceedings of an International Workshop*, pages 269–280, 1981.

M.M. Sigurd, N.L. Ulstein, B. Nygreen, and D.M. Ryan. Ship scheduling with recurring visits and visit separation requirements. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, pages 225–245. Springer US, 2005.

M. Stålhane, H. Andersson, M. Christiansen, J.-F. Cordeau, and G. Desaulniers. A branch-price-and-cut method for a ship routing and scheduling problem with split loads. *Computers and Operations Research*, 39(12):3361–3375, 2012.

M. Stålhane, H. Andersson, M. Christiansen, and K. Fagerholt. Vendor managed inventory in tramp shipping. *Omega (United Kingdom)*, 47:60–72, 2014.

UNCTAD. Review of maritime transport 2013. `http://unctad.org/en/PublicationsLibrary/rmt2013_en.pdf`, 2013.

C. Vilhelmsen, R. Lusby, and J. Larsen. Tramp ship routing and scheduling with integrated bunker optimization. *EURO Journal on Transportation and Logistics*, 2013. doi: 10.1007/s13676-013-0039-8.