OR Spectrum (2017) 39:913–943
DOI 10.1007/s00291-017-0480-4

REGULAR ARTICLE

Tramp ship routing and scheduling with voyage separation requirements

Charlotte Vilhelmsen[1] · **Richard M. Lusby**[1] · **Jesper Larsen**[1]

Received: 29 November 2016 / Accepted: 21 April 2017 / Published online: 3 May 2017

Abstract In this paper we explore tramp ship routing and scheduling. Tramp ships operate much like taxies following the available demand. 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 describe a time window branching scheme used to enforce the voyage separation requirements which we relax in the master problem. Computational results show that our algorithm in general finds optimal solutions very quickly and performs much faster compared to an earlier a priori path generation method. Finally, we compare our method to an earlier adaptive large neighbourhood search heuristic and find that on similar-sized instances our

✉ Jesper Larsen
jesla@dtu.dk

Charlotte Vilhelmsen
chaan@dtu.dk

Richard M. Lusby
rmlu@dtu.dk

1 Department of Management Engineering, Technical University of Denmark Produktionstorvet, Building 426, 2800 Kongens Lyngby, Denmark

approach generally uses less time to find the optimal solution than the adaptive large neighbourhood search method uses to find a heuristic solution.

Keywords Scheduling · Maritime transport · Spread requirement · Optimization · Exact algorithms

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. 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 and ongoing problem is how to most efficiently route and schedule their fleets according to current demand. This is precisely the problem we consider. However, we add further complexity to the problem by incorporating voyage separation requirements (VSRs) which enforce a minimum time spread between some voyages. This is done in order to ensure that similar voyages are performed fairly evenly spread in time. Such an even time spread is a requirement in some shipping contracts in order to reduce inventory costs for the charterer. In addition to lowering inventory costs for the charterer, a more even distribution of similar voyages also increases the likelihood that the ship operator will be able to find sufficient cargoes in the market to fill the ship on each voyage while not finding more cargoes than ship capacity allows.

Norstad et al. (2015) describe a similar problem which originates from a Norwegian shipping company called Saga Forest Carriers that is difficult to categorise as either a liner, tramp or industrial ship operator. Saga Forest Carriers transport forest products as well as break bulk cargoes world-wide on fixed routes on a regular basis; however, they also take on additional voyages in the spot market. We view the problem from a tramp shipping perspective as a routing and scheduling problem.

Results from Norstad et al. (2015) show that their methods struggle on larger and more complex problem instances. Since then Saga Forest Carriers have expanded its fleet and business from the 25 vessels considered by Norstad et al. (2015) to 32 vessels. Bakkehaug et al. (2016) report that the best method from Norstad et al. (2015) runs out of memory on the larger instances and have therefore presented an adaptive large neighbourhood search (ALNS) heuristic for the same problem. They report good quality solutions within relatively short computation times. The aim here is therefore to develop an exact method that is more efficient than the a priori path generation method from Norstad et al. (2015).

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 (BAP) procedure with a dynamic programming algorithm to generate the columns. A time window branching scheme is described and used to enforce the voyage separation requirements, which are relaxed in the master problem.

The remainder of the paper is organised as follows. In Sect. 2 we review relevant literature while in Sect. 3 we provide a problem description as well as a nonlinear mathematical model for the problem. In Sect. 4 we describe the decomposition of the problem and the dynamic column generation procedure, i.e. the pricing part of the proposed algorithm, and in Sect. 5 we describe the branching part of the algorithm. In Sect. 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 Norstad et al. (2015) and those presented in Bakkehaug et al. (2016). Finally, concluding remarks and suggestions for future work are discussed in Sect. 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 reviews Ronen (1983), Ronen (1993), Christiansen et al. (2004) and Christiansen et al. (2013), while Vilhelmsen et al. (2015) review more recent literature on tramp ship routing and scheduling. The problem considered here is closely related to the vehicle routing problem with time windows, for which we refer the reader to Desaulniers et al. (2002).

In this paper we extend the tramp ship routing and scheduling problem by including voyage separation requirements. Such requirements are, as already mentioned, also considered in Norstad et al. (2015) in a context similar to ours. They present an arc flow formulation that is solved directly using commercial software. They also present a path flow formulation, which is solved by a priori column generation and a commercial solver for the final problem. In their path flow formulation each column corresponds to a geographical route for which the timing of the included port calls is determined in the path flow formulation through the inclusion of time variables and related time constraints. The computational results from Norstad et al. (2015) show that both their formulations can be used to solve smaller instances, while the path flow formulation can also be used to solve problems of more realistic sizes. However, neither of these methods are applicable for larger and more complex problem instances.

Since the two exact methods presented in Norstad et al. (2015) struggle on larger and more complex problem instances Bakkehaug et al. (2016), as already mentioned, take a heuristic approach to the same problem. They propose an adaptive large neighbourhood search heuristic with paired destroy and repair operators and normalised scores for these pairs. They compare their proposed heuristic to the path flow formulation from Norstad et al. (2015) and find that their heuristic yields good quality solutions within relatively short computation times.

Stålhane et al. (2014) also relate tramp ship routing and scheduling to the broader context of the supply chain. They do this by combining traditional tramp shipping with a vendor managed inventory (VMI) service in an attempt to challenge the traditional *Contracts of Affreightment* which is the standard agreement between a tramp ship company and a charterer. The authors present an arc flow formulation for this problem as well as a path flow formulation. The path flow formulation is solved by a hybrid method that combines a priori path generation of all feasible routes with BAP to generate schedules for these routes. Larger instances are solved using a heuristic version of path generation. Computational results show that the profit and efficiency of the supply chain can be significantly increased by using vendor managed inventory services compared to the traditional contracts of affreightment. Computational results also show that the heuristic can significantly speed up computation time compared to the exact method, and at only a small reduction in solution quality. However, even with the heuristic approach they are only able to solve small-sized instances where only few VMI services are introduced, and running times increase drastically from seconds to days with increases in problem size or in the number of VMIs.

Hemmati et al. (2015) continue the work from Stålhane et al. (2014) in order to develop a more efficient method for the same problem, i.e. for a tramp ship routing and scheduling problem with inventory constraints. The authors present a new heuristic method that works in two phases by first converting the inventory constraints into cargoes, and then solving the resulting problem with an adaptive large neighbourhood search heuristic. This procedure continues iteratively by changing the cargoes derived from the inventory constraints and resolving the resulting tramp ship routing and scheduling problem. Computational results show that the heuristic can solve realistically sized instances of the tramp ship routing and scheduling problem with inventory constraints in reasonable time, and that when the number of inventory pairs is large, the heuristic is much faster than the methods from Stålhane et al. (2014) and generally finds better solutions.

When incorporating the inventory aspect into the tramp ship routing and scheduling problem, it is necessary to include some form of temporal dependency between ship schedules that perform similar voyages. However, we can find numerous examples of other applications that require the use of various kinds of temporal dependencies in routing and scheduling problems. One such application within tramp shipping can be found in Andersson et al. (2011) that consider a special tramp segment called project shipping where cargoes are more unique and usually transported on a one-time basis, e.g. cable reels, used school buses, other vessels and such ’odd’ commodities. If some cargoes are part of a larger project, they might be required to be delivered almost simultaneously whereby synchronisation constraints arise. The authors present an arc flow formulation and suggest three different solution methods that all rely on a priori generation of all feasible routes. Computational results show that they are able to solve real-life-sized planning problems. Stålhane et al. (2015) present a new mathematical formulation for the problem and develop a BAP method with a new variant of the elementary shortest path problem with resource constraints solved through dynamic programming. Their computational results show that this new method is indeed more efficient than the ones from Andersson et al. (2011).

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. A heuristic BAP algorithm is used 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. Synchronisation constraints are often encountered in vehicle routing, see e.g. Reinhardt et al. (2013) who consider synchronised 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 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. Computational results show 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 generalised 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, where a voyage is a sailing through a set of ordered ports to first pick up cargo and afterwards discharge it. The operator can however 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 maximising 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 characterised 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. A ship can only perform one voyage at a time.

Several voyages can be identical except for their time windows for start of service. In fact, contract voyages 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*. Hence, 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.

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 3 weeks. As discussed in Norstad et al. (2015), 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. (2015).

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 characterised 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 each ship can only perform one voyage at a time we assume that each ship is approximately fully loaded during each voyage. Thereby, 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. In addition, contracts of affreightments define the sailing speed in order to make sure that departure and arrival on a voyage are well defined between tramp operator and contract owner. Speed optimisation is therefore not part of this problem. 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 $\mathcal{I}_r = \{1, 2, \dots, n_r\}$ 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 modelled 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 or must perform maintenance stop (r, i). 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_{riqj}^k 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_{riqj}^k 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 $T_{o(k)ri}^k$ denote the time for travelling ballast with ship k from its origin to the first port of voyage or maintenance (r, i).

For the ballast legs, C_{riqj}^k and $C_{o(k)ri}^k$ denote, respectively, the cost of travelling 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_{ri}^k, 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_{ri}^S.

For the mathematical formulation we need several variables. First, we define binary flow variables x_{riqj}^k 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 $x_{o(k)ri}^k$, $x_{rid(k)}^k$ and $x_{o(k)d(k)}^k$ 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 $t_{o(k)}^k$ and t_{ri}^k 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_{ri}^S and let the binary variable y_{ri} be equal to 1.

We can now give an arc flow formulation of the TSRSPVSR similar to the one presented in Norstad et al. (2015):

$$\max \sum_{k\in\mathcal{V}} \sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} \sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} \left(P_{ri}^k - C_{riqj}^k\right) x_{riqj}^k + \sum_{k\in\mathcal{V}} \sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} P_{ri}^k x_{rid(k)}^k - \sum_{k\in\mathcal{V}} \sum_{r\in\mathcal{R}^k} \sum_{i\in\mathcal{I}_r} C_{o(k)ri}^k x_{o(k)ri}^k - \sum_{(r,i)\in\mathcal{N}_C} C_{ri}^S y_{ri} \quad (1)$$

s.t.

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

$$\sum_{k\in\mathcal{V}_r} \left(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x_{riqj}^k + x_{rid(k)}^k \right) \le 1, \qquad \forall (r,i) \in \mathcal{N}_O, \quad (3)$$

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

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

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

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

$$x_{o(k)qj}^k (t_{o(k)}^k + T_{o(k)qj}^k - t_{qj}^k) \le 0, \qquad \forall k \in \mathcal{V},\ q \in \mathcal{R}^k,\ j \in \mathcal{I}_q, \quad (8)$$

$$x_{riqj}^k (t_{ri}^k + T_{riqj}^k - t_{qj}^k) \le 0, \qquad \forall k \in \mathcal{V},\ ((r,i),(q,j)) \in \mathcal{A}^k, \quad (9)$$

$$a_{ri}\big(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x_{riqj}^k + x_{rid(k)}^k \big) \le t_{ri}^k \le b_{ri} \left(\sum_{q\in\mathcal{R}^k} \sum_{j\in\mathcal{I}_q} x_{riqj}^k + x_{rid(k)}^k \right), \qquad \forall k \in \mathcal{V}, r \in \mathcal{R}^k,\ i \in \mathcal{I}_r, \quad (10)$$

$$\sum_{k\in\mathcal{V}_r} t_{ri}^k + t_{ri}^S y_{ri} + B_r \le \sum_{k\in\mathcal{V}_r} t_{r,i+1}^k + t_{r,i+1}^S y_{r,i+1}, \qquad \forall (r,i) \in \mathcal{N}_C, i \in \mathcal{I}_r \backslash \{n_r\}, \quad (11)$$

$$a_{ri} \le t_{ri}^S \le b_{ri}, \qquad \forall (r,i) \in \mathcal{N}_C, \quad (12)$$

$$t_{o(k)}^k \ge a_{o(k)}, \qquad \forall k \in \mathcal{V}, \quad (13)$$

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

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

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

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

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

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 service time as well as sailing time is respected from the origin to the first node on the itinerary. 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 BAP scheme later.

4 Decomposition

The nonlinear mixed integer programming model (1)–(18) could in theory be solved by commercial optimisation software for linear problems after linearising constraints (8), (9), and (11). However, as pointed out in Norstad et al. (2015) 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.

For each ship, if the same voyage and maintenance stops can be ordered into numerous different feasible geographical routes, only the route with the highest profit would have to be included in the master problem if the VSR constraints were not considered. 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. Hence, with the VSR constraints included in the problem the master problem column set can contain several columns all corresponding to the same set of voyage and maintenance stops. In this case a priori generating all columns can be very time consuming and also result in large master problems.

If a schedule contains waiting time, we can redistribute the waiting time and obtain a different schedule corresponding to the same ship and geographical route. So 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. However, with the VSR constraints included in the problem the master problem column set can contain several columns all corresponding to the same set of voyage and maintenance stops and even to the same geographical route. A priori generating all columns will therefore result in a tremendous amount of columns depending on the time discretisation used to redistribute the waiting time.

Norstad et al. (2015) approach this difficulty by including time variables in the master problem so that each column correspond to a specific geographical route, or a *path* in the underlying network. This way only one column needs to be included for each geographical route though at the obvious expence of creating a more complex master problem due to the additional time variables and related constraints. At the same time they might still need to include several columns corresponding to the same set of voyage and maintenance stops.

Therefore, we turn to dynamic column generation (see e.g. Desaulniers et al. (2005) for a general description) 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 BAP 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.

We 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. 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 (r, i) in schedule s, and 0 otherwise. Finally, we denote the start time for voyage (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.

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

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

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

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

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

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

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

We 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 Sect. 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.

The entire BAP process therefore begins with the solution of 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 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'. 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.

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 $(r,i) \in \mathcal{N}^k_M$ and let ω_k be the dual for constraint (22) which is also free of sign. Since we consider the generic subproblem we drop the superscript k on the variables and the subproblem is then given by:

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

s.t.

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

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 modelled as an elementary shortest path problem with resource constraints (ESPPRC) which is $\mathcal{NP}$-hard in the strong sense (see Dror 1994).

The ESPPRC is also known to be a hard problem to solve from a practical point-of-view. The related shortest path problem with resource constraints (SPPRC) allows for multiple visits of nodes and is a relaxation of the ESPPRC. The SPPRC is known to have a pseudo- polynomial time complexity (see e.g. Desrochers and Soumis 1988, Irnich and Desaulniers 2005). This should be seen in relation to the subproblem we need to solve. 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 SPPRC. In Sect. 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 Irnich 2008 and Irnich and Desaulniers 2005 for a thorough introduction to the SPPRC.

4.2.1 Subproblem networks

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.

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 $T_{n_1 n_2}$, $C_{n_1 n_2}$ and $M_{n_1 n_2}$, 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. 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 and correspondingly for the time consumption. 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 travelling ballast from the maintenance port corresponding to n_1 to the first port of the voyage corresponding to n_2.

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.

4.2.2 Dynamic programming algorithm

Given a dual solution to an optimised 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. Updating the cost $\hat{C}_{n_1,n_2}$ on arc (n_1, n_2) corresponds to:

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

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

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

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.

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 Sect. 4.3 we discuss what to do with these schedules. See Algorithm 1 for a general overview of our label setting algorithm.

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 $\mathcal{S}$
1  Sorted Node List $\hat{\mathcal{N}} \leftarrow$ topologicalSort(G);
2  CreateInitialLabel($o$);
3  for $u \in \hat{\mathcal{N}}$ and $u \neq d$ do
4      $\mathcal{L}_u \leftarrow$ getLabels(u);
5      for $l \in \mathcal{L}_u$ do
6          if $l$ is not dominated then
7              for $a \in$ outgoingArcs($u$) do
8                  if extension($l, a$) is feasible then
9                      $v \leftarrow$ headNode($a$);
10                     createLabel($l, a, v$);
11                     $\mathcal{L}_v \leftarrow$ getLabels($v$);
12                     dominanceCheck($\mathcal{L}_v$);
13                 end
14             end
15         end
16     end
17 end
18 $\mathcal{L}_d \leftarrow$ getLabels($d$);
19 $\mathcal{S} \leftarrow$ constructSchedules($\mathcal{L}_d$);
20 return $\mathcal{S}$;
```

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. Through tuning and testing the algorithm we have verified 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.

Each Pareto optimal schedule returned to the master problem can potentially be converted into numerous new schedules with the same route but with slightly different schedules if we simply redistribute any waiting time along the route. We have implemented such a waiting time redistribution routine and tested whether it would be beneficial to include such additional schedules in the master problem. This redistribution routine did not improve the results from the algorithm; rather it seemed to flood the master problem with irrelevant columns causing excessive time usage for solving the RMP. Therefore, the redistribution routine is not used in the final version of the algorithm.

Through tuning and testing we found that for this problem it is most efficient to solve one subproblem in each iteration and then solve the correspondingly updated RMP to obtain new dual values before we solve the subproblem for the next ship.

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 fulfils 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). Furthermore, we extend their methods to also account for the spot vessel time window constraints (24) and use a 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. We will in the rest of this section focus on the implementation of time window branching and VSR violations. For the transfer of the "basic" time window branching scheme to the TSRSVSR problem we have largely omitted the technical details and refer to Vilhelmsen (2014).

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

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	$[a_{ri}, \min\{b_{ri}, b_{r,i+1} - B_r\}]$	$[\max\{a_{r,i+1}, a_{ri} + B_r\}, b_{r,i+1}]$

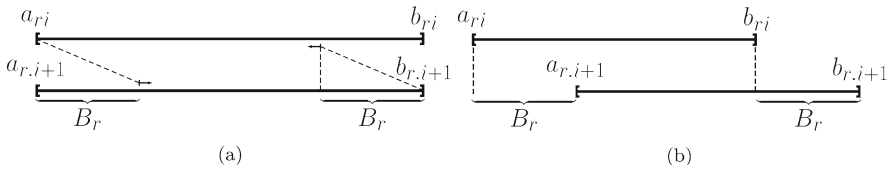

Fig. 1 Time window reduction process for two consecutive nodes on a trade. **a** Before reduction, and **b** after reduction

reductions for two consecutive nodes (r, i) and $(r, i + 1)$ on trade r. The reduction process is illustrated in Fig. 1.

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.

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 fulfil 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. We abuse notation slightly by letting T_i^1 and T_i^2 denote the visit time for these two visits, respectively. 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 Fig. 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. A formal description and a proof for viability 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.

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. All 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 that the VSR constraint for two consecutive voyages, i and $i + 1$, on trade r is violated and that 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

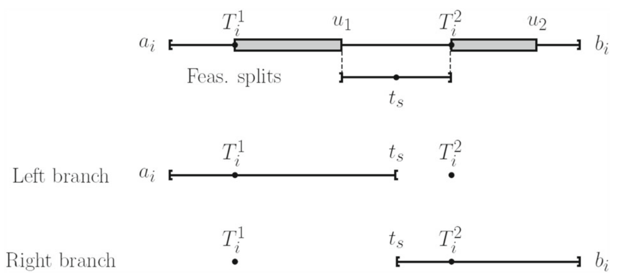

Fig. 2 Time window branching due to fractionality

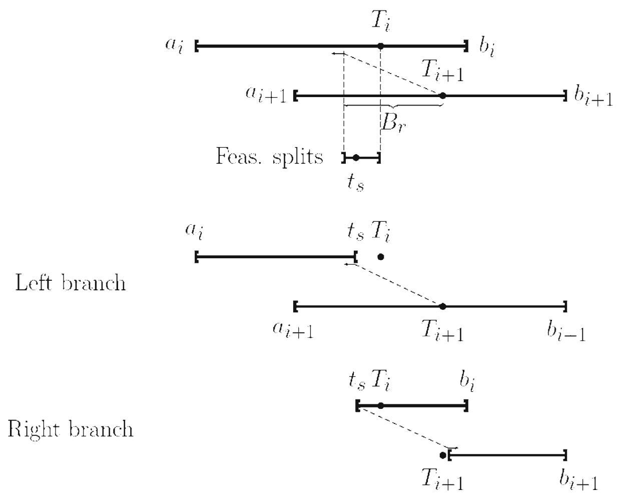

Fig. 3 Time window branching due to VSR violation

that $T^{k_1}_{ris_1} + B_r > T^{k_2}_{r,i+1,s_2}$. Again we let 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. 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).

We 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:

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

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

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 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 put together and forms an LP for this particular *VSR group*.

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

$$\exists g \in \{0, 1, \ldots, m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + B_r > T^{k_2}_{r,i-g+1,s_2}, \tag{36}$$

$$\exists g \in \{1, 2, \ldots, m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + 2B_r > T^{k_2}_{r,i-g+2,s_2}, \tag{37}$$

$$\exists g \in \{2, 3, \ldots, m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + 3B_r > T^{k_2}_{r,i-g+3,s_2}, \tag{38}$$

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

$$\exists g \in \{m-1, m\}, s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-g,s_1} + mB_r > T^{k_2}_{r,i-g+m,s_2}, \tag{39}$$

$$\exists s_1, s_2, k_1, k_2 : \lambda^{k_1}_{s_1} > 0, \lambda^{k_2}_{s_2} > 0, T^{k_1}_{r,i-m,s_1} + (m+1)B_r > T^{k_2}_{r,i+1,s_2}. \tag{40}$$

To exemplify, assume that the current solution has $y_{r,i-1}$ and y_{ri} positive while $y_{r,i-2} = y_{r,i+1} = 0$. We then 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:

$$T^{k_1}_{r,i-2,s_1} + B_r > T^{k_2}_{r,i-1,s_2} \tag{41}$$

$$T^{k_1}_{r,i-1,s_1} + B_r > T^{k_2}_{ris_2} \tag{42}$$

$$T^{k_1}_{ris_1} + B_r > T^{k_2}_{r,i+1,s_2} \tag{43}$$

$$T^{k_1}_{r,i-2,s_1} + 2B_r > T^{k_2}_{ris_2} \tag{44}$$

$$T^{k_1}_{r,i-1,s_1} + 2B_r > T^{k_2}_{r,i+1,s_2} \tag{45}$$

$$T^{k_1}_{r,i-2,s_1} + 3B_r > T^{k_2}_{r,i+1,s_2} \tag{46}$$

Constraint (43) corresponds to the illustration in Fig. 3 and this figure must now be extended to include the time windows and visit times corresponding to constraints (42) and (44). The remaining violations will be handled when branching on nodes $(r, i-2)$, $(r, i-1)$ and $(r, i+1)$.

We run through each VSR group and only for the violated ones, we search for visit times that fulfil constraints (36)–(40), starting from (36). If we find one or several pairs of visit times that fulfil constraints (36), we have one or several pairs of candidate nodes for branching. In that case we do not check constraints (37)–(40) since these can be implicitly handled by branching to fulfil constraints (36) and using the time window reduction rule. However, if we do not find a pair of visit times that fulfil constraints (36), we move on to check constraints (37) and so the process continues until we find branching candidates.

5.1.3 Selecting the best split time within a time window

Assume for now that the current solution is fractional but fulfils 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. A detailed description on how to transfer this to the TSPSPVSR without considering the VSR violations we refer to Vilhelmsen (2014).

We will here and now concentrate on how we include the VSR violations in the time window branching scheme.

We assume that nodes $(r, i-1)$, (r, i) and $(r, i+1)$ together form a VSR group. Figure 4 includes 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^1_{i-1} and T^1_{i+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 Fig. 4. 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 Fig. 4, 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, $T^1_{i+1} - B_r$ and $T^1_{i-1} + B_r$. 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, $T^1_{i-1} + B_r$ 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

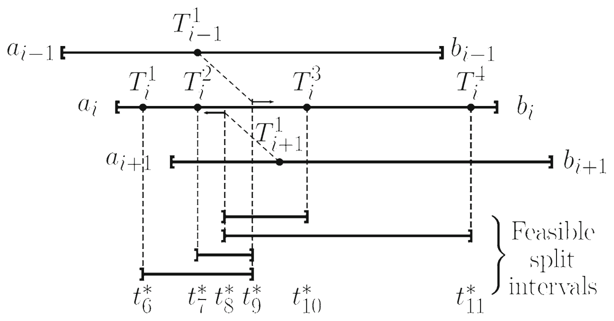

Fig. 4 Choosing a split time to restore VSR

Table 2 Flow elimination from candidate split times

Candidate	Infeasible visits		Eliminated flow		Minimum
	Left branch	Right branch	Left branch	Right branch	
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

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.

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 Fig. 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 Fig. 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. Therefore, when branching on a node where there are no VSR violations, we reinclude the feasibility interval aspect to more effectively eliminate fractionality.

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.

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.

5.2 Constraint branching

For now, ignore the possible existence of cyclic master problem schedules, i.e. columns with $A_{ris} > 1$ in (19)–(26). 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). Constraint branching is an effective technique for enforcing integrality of problems with such structure and was first proposed in Ryan and Foster (1981). The authors observed that in an optimal integer solution to a set partitioning problem any two rows are either covered by the same variable or covered by two different variables. In a fractional solution this is not the case and two branches can be enforced to eliminate this fractionality. The first restricts the solution space by enforcing the requirement that the two rows be covered by the same variable, while the second ensures that they are not covered by the same variable. Such a branching strategy clearly partitions the solution space into two disjoint subspaces. Due to the generalised upper bound constraints (22), the submatrix for each ship in the RMP is perfect, see e.g Padberg (1973) and Conforti et al. (2001). As a result, 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. In a fractional solution there must also exist two voyages that are performed consecutively in one schedule and not consecutively in another. This is known as a follow on branching possibility (see e.g. Steinzen et al. 2009), since we enforce/restrict what voyage is allowed to immediately follow another voyage.

If the current solution is fractional, there must exist two voyages $(r_1, i_1), (r_2, i_2) \in \mathcal{N}_C \cup \mathcal{N}_O$ with $i_1 \neq i_2$ that are performed consecutively by some ships and nonconsecutively by other ships. For each such pair we introduce the sum

$$S_{(r_1,i_1),(r_2,i_2)} = \sum_{k \in \mathcal{V}} \sum_{s \in \mathcal{S}^{k'}} A^k_{r_1 i_1 s} A^k_{r_2 i_2 s} \lambda^k_s.$$

If the current solution is fractional, there must exist consecutive voyages (r_1, i_1) and (r_2, i_2) for which $0 < S_{(r_1,i_1),(r_2,i_2)} < 1$. The branching strategy is then to construct a left branch where voyage (r_1, i_1) is immediately followed by voyage (r_2, i_2) (termed the 1-branch) and a right branch where voyage (r_2, i_2) is not allowed to immediately follow voyage (r_1, i_1) (termed the 0-branch).

The approach for selecting the branching pair varies in the literature as basically two different opinions on this selection exist: Selecting the (voyage, voyage) pair with largest $S_{(r_1,i_1),(r_2,i_2)}$ and selecting the (voyage, voyage) pair with the most fractional $S_{(r_1,i_1),(r_2,i_2)}$. We have implemented both of these versions as well as a hybrid version that combines them. We found the most fractional approach most effective for the problem considered here.

$$((r_1, i_1), (r_2, i_2))^* = \arg \min_{((r_1,i_1),(r_2,i_2)) \in \mathcal{C}} \left\{ \left| S_{(r_1,i_1),(r_2,i_2)} - \frac{1}{2} \right| \right\}.$$

Returning to the possible existence of cyclic master problem schedules, we note that their presence in the master problem constraint matrix will compromise the perfectness of the submatrices containing such schedules. The above constraint branching scheme cannot eliminate fractionality derived from cyclic schedules and we must instead rely on time window branching for this.

5.3 Branching strategy

We have implemented a breadth-first search and found the branching most effective when we prioritised the VSR-related time window branching scheme. Therefore, we start 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 and that time window branching must be used if fractionality occurs due to cyclic schedules.

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 provide a comparison of the proposed BAP algorithm with that of the a priori path generation method from Norstad et al. (2015) on a subset of the considered instances.

6.1 Data instances

The BAP algorithm is tested on 19 problem instances of varying size and complexity. These are divided into two sets. The first, containing 14 instances, is a set of instances for which we can compare the results to those obtained using the method of Norstad

Table 3 Data instance characteristics

No.	Ships	Trades	Voyages	Spot	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	44	11	90
11	25	8	53	11	90
12	25	8	57	10	105
13	25	8	55	5	105
14	25	9	64	4	120
15	25	8	49	10	90
16	25	9	58	4	105
17	25	8	64	12	120
18	25	8	62	9	120
19	32	13	56	12	90

et al. (2015). The second is a smaller set, containing 5 instances, on which we further test our methodology. No comparison is possible for these instances. All instances have been generated by the test instance generator described in Norstad et al. (2015). This is based on data from the Norwegian shipping company Saga Forest Carriers. Table 3 presents the main characteristics of the 19 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, the number of spot voyages, and 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. Also note that instance 19 is the only instance that includes maintenance requirements. For this instance there is one maintenance requirement corresponding to one of the 13 trades stated in Table 3.

6.2 Computational results

In order to evaluate the performance of the devised algorithm, we run it on the 19 data instances, and compare the results obtained from these tests on a subset of the considered instances with results from using the a priori path generation method from Norstad et al. (2015).

6.2.1 Results from devised BAP method

The proposed BAP method is programmed in C++ and all experiments have been performed on a dedicated Intel(R) Xeon(R) CPU X5550 @ 2.67GHz with 24 gigabytes of main memory running Ubuntu Linux version 14.04. We use default settings for the commercial solver Cplex 12.4 to solve the master problem. We tried different values for the time window branching parameter ϵ and found only slight changes in running time of the algorithm. High ϵ values can potentially sacrifice optimality; however, the quality of the solutions did not change. We therefore chose a small ϵ value, namely 0.05 which corresponds to 72 min on our data instances compared to the very long voyage time windows of up to 30 days.

Tables 4 and 5 contain the results for two different versions of the BAP method on the 19 available instances. In particular, we investigate the impact of the fractional time window branching routine. Table 4 contains the results where fractional time window branching is not used, while Table 5 presents the results when all three branching strategies are used. Looking at both tables, the column 'Inst' gives the instance number and column 'Gap' gives the integrality gap (as a percentage) for the best found solution found. In column 'Time' we list the time used to solve the problem (in seconds). We permit the algorithm to run for at most 600 s. The number of nodes evaluated in the BAP tree is given in the column 'Nodes', while the column 'Vars' lists the number of variables in the final master problem. Finally, in the last three columns we report the number of times each of the branching strategies were applied. Here, 'fTW' refers to fractional time window branching, 'iTW' refers to integral time window branching, and 'FB' refers to follow on branching.

From Table 4, we see that the version of the BAP algorithm in which fractional time window branching is not used solves all instances to optimality very quickly. Only two instances (17 and 18) require more than 20 s of computation time. Furthermore, the number of columns generated is never more than 2200 and we investigate, in the worst case, approximately 37,000 nodes. The performance of the algorithm is satisfactory without fractional time window branching. Table 5 provides an overview of what happens when this is also included. The results are mixed. On the one hand we see a substantial reduction in the running times for several instances (e.g. 11, 16, and 18); however, we also see a dramatic increase for others. In two cases, instances 13 and 17, the algorithm is now unable to prove optimality within the 10 min time frame. In both cases the integrality gap is, however, small and the best found solutions are in fact optimal. In both cases the best solution were found within 14 s. Due to the fact that two instances time out, we see larger BAP trees and more columns generated. A possible explanation for the large BAP trees, and ultimately the time out, could be that time window branching can create schedules that differ in the timing of port visits by very small amounts of time, and this is more pronounced when fractional time window branching is also included. The results do, however, suggest that fractional time window branching has potential. On instance 18 in particular, the running time is reduced by a factor of 10 when fractional time window branching is included. When we compare our BAP algorithm with the a priori path generation method from Norstad et al. (2015) in Sect. 6.2.2 we use the version of the BAP algorithm in which only integral time window branching and follow on branching is used. Finally, we note that,

Table 4 Results from BAP method (no fTW)

Inst	Gap	Nodes	Vars	Time	iTW	FB
1	–	45	156	0.04	4	18
2	–	15	219	0.06	0	10
3	–	13	157	0.04	0	6
4	–	49	303	0.06	6	18
5	–	3	262	0.02	0	1
6	–	32	410	0.08	12	13
7	–	4	355	0.04	3	0
8	–	1	262	0.02	0	0
9	–	97	396	0.21	1	47
10	–	56	431	0.16	25	8
11	–	4229	1052	7.92	1168	1410
12	–	30	599	0.18	13	15
13	–	1609	1089	3.83	267	537
14	–	3038	1581	19.42	561	2435
15	–	542	584	1.13	143	344
16	–	2163	1219	8.30	402	1372
17	–	37,053	2133	198.89	2866	15, 660
18	–	10,061	1564	51.28	703	4327
19	–	185	681	0.63	7	85

in both Tables 4 and 5, there does not really seem to be any trend in the distribution of the applied branching schemes.

As already mentioned, Norstad et al. (2015) 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 all instances again without voyage separation requirements (using the BAP approach without fractional time window branching). In fact, the profit reduction is 0% on most instances while instance 14 experiences the highest profit reduction, and this is only 0.19%. The complexity added from the voyage separation requirements is however not insignificant. In fact, the running time of the algorithm is reduced by 0–99.9% where the latter is for instance 17.

6.2.2 Comparing the two methods

To properly evaluate the efficiency of our devised BAP algorithm (without fractional time window branching) we use this section to compare our results with those obtained using the A Priori Path Generation (APPG) method described in Norstad et al. (2015). The APPG method first generates all feasible *paths* and then uses a commercial solver to solve the path flow formulation containing the set of generated paths. Note that the enumeration concerns paths and not schedules; i.e. routes are enumerated while the timing of these routes is left as decision variables in the path formulation.

Table 5 Results from BAP method (with fTW)

Inst	Gap	Nodes	Vars	Time	fTW	iTW	FB
1	–	91	182	0.06	5	1	39
2	–	15	219	0.06	0	0	10
3	–	13	157	0.04	0	0	6
4	–	71	374	0.06	5	3	27
5	–	7	285	0.03	2	1	2
6	–	32	410	0.08	0	12	13
7	–	4	355	0.04	0	3	0
8	–	1	262	0.02	0	0	0
9	–	503	141	0.41	14	3	53
10	–	56	447	0.15	31	6	9
11	–	606	908	2.00	515	87	3
12	–	5	618	0.16	4	0	0
13	0.19	69, 379	3337	600.00	28, 523	188	25, 088
14	–	3305	2045	14.53	1590	95	120
15	–	230	714	0.64	195	29	4
16	–	606	908	2.37	318	56	23
17	0.02	60, 056	3131	600.00	8695	56	30, 676
18	–	871	1448	5.40	156	20	259
19	–	215	686	0.70	4	8	95

The authors from (Norstad et al. 2015) have provided us with results from running their APPG method on the first 14 data instances described in this paper. Their results are obtained using a Dell Latitude Laptop with Intel Core i5 CPU (4 $\times$ 2.40 GHz), 4GB DDR2 RM running on Windows 7. The authors' path generator is implemented in C#, while the path flow model is solved with Xpress MP 7.0. Their results are given in Table 6, along with the key values from Table 4. The column headers are the same as in Table 4 and for instance 14 the '*' indicates that the APPG method exceeded the time limit of 3600 s before closing the integrality gap.

From Table 6 we first note that the time required by the APPG method is generally a lot longer than that required by the proposed algorithm. We acknowledge that the processor used in the experiments of Norstad et al. (2015) is slightly slower than the one we use; however, the differences in running times are substantial and therefore cannot be attributed to this.

There is a significant reduction in model size from relaxing the VSRs and using dynamic column generation compared to including the VSRs in the model and using a priori column generation as in the APPG method. We note that the BAP approach consistently includes much fewer variables than the APPG method; in fact, the variable count in the BAP approach is 84–98% lower than that of the APPG method. This is the advantage when using dynamic column generation; the method only generates columns that have the potential to improve the objective function value. We do remind

Table 6 Comparing solution methods

Inst	APPG		BAP	
	Gap	Time	Gap	Time
1	–	0	–	0
2	–	1	–	0
3	–	0	–	0
4	–	15	–	0
5	–	1	–	0
6	–	2	–	0
7	–	2	–	0
8	–	3	–	0
9	–	36	–	4
10	–	2	–	0
11	–	19	–	3
12	–	14	–	0
13	–	152	–	4
14	0.13	3600*	–	19

the reader, however, that the APPG method does not enumerate all feasible schedules, but instead all feasible paths/routes. Similarly, the number of constraints explicitly included in the APPG model is far greater than that for the proposed model; in fact, the number of constraints with the BAP approach is 99.4–99.9% lower than with the APPG method. Again, this is to be expected since we relax the voyage separation constraints and also do not include any constraints on timing since these are implicitly included in the column generation subproblems. Overall, we find that the devised BAP algorithm is superior to that of the APPG method, providing optimal solutions much faster. Instances 13 and 14 in Table 6 show that the difference in run time can be quite dramatic. For instance 13, the BAP method proves optimality within 4 s, while the APPG takes approximately 2.5 min. For instance 14, the BAP method proves optimality within 20 s, while the APPG cannot prove optimality within an hour of computation time.

As mentioned in Sect. 2 Bakkehaug et al. (2016) present results from using an adaptive large neighbourhood search (ALNS) heuristic for the same problem. We note that, except for instances 17 and 18 (in this paper), on similar-sized instances our BAP approach uses less time to find the optimal solution than the ALNS method uses to find a heuristic solution and that on average their heuristic solutions exhibit a 0.69% optimality gap which is approximately $ 200,000. On larger instances the comparison in Bakkehaug et al. (2016) with Norstad et al. (2015) reveals gaps above 1%.

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 ensure that the voyages are ‘fairly evenly spread’ in time. A more evenly distribution of similar voyages also increases the likelihood that the ship operator will be able to find sufficient cargoes in the market to fill the ship on each voyage while not finding more cargoes than ship capacity allows.

We have developed a new, exact method for this problem. It is a BAP 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.

Running our algorithm on all 19 instances without the voyage separation requirements showed that the profit reduction from including the separation requirements is below 0.2% on all instances.

We compared our BAP method to the APPG method from Norstad et al. (2015) and found that the time usage from the APPG method is generally a lot longer than that for our algorithm. Furthermore, we compared our BAP method to an ALNS heuristic from Bakkehaug et al. (2016) and found that on similar-sized instances our BAP approach generally uses less time to find the optimal solution than the ALNS method uses to find a heuristic solution and that on average their heuristic solutions exhibit a 0.69% optimality gap.

Overall we have developed a new, exact and efficient method for the tramp ship routing and scheduling problem with voyage separation requirements. This method is very fast at finding optimal solutions, although one instance requires a bit longer time.

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

Andersson H, Duesund JM, Fagerholt K (2011) Ship routing and scheduling with cargo coupling and synchronization constraints. Comput Ind Eng 61(4):1107–1116

Bakkehaug R, Rakke JG, Fagerholt K, Laporte G (2016) An adaptive large neighborhood search heuristic for fleet deployment problems with voyage separation requirements. Transp Res C Emerg Technol 70:129–141

Christiansen M, Fagerholt K, Ronen D (2004) Ship routing and scheduling: status and perspectives. Transp Sci 38(1):1–18

Christiansen M, Fagerholt K, Nygreen B, Ronen D (2007) Maritime transportation. In: Barnhart C, Laporte G (eds) Transport. Handbooks in operations research and management science, vol 14. Elsevier, North-Holland, Amsterdam, pp 189–284

Christiansen M, Fagerholt K, Nygreen B, Ronen D (2013) Ship routing and scheduling in the new millennium. Eur J Oper Res 228(3):467–483

Conforti M, Cornuéjols G, Kapoor A, Vušković K (2001) Perfect, ideal and balanced matrices. Eur J Oper Res 133(3):455–461

Cordeau J-F, Desaulniers G, Desrosiers J, Solomon MM, Soumis F (2002) VRP with time windows. In: Toth P, Vigo D (eds) The vehicle routing problem, chap. Society for Industrial and Applied Mathematics, Philadelphia, pp 157–194

Desaulniers G, Desrosiers J, Solomon MM (eds) (2005) Column generation. Springer, New York

Desrochers M, Soumis F (1988) A reoptimization algorithm for the shortest path problem with time windows. Eur J Oper Res 35:242–254

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

Drexl M (2013) Applications of the vehicle routing problem with trailers and transshipments. Eur J Oper Res 227(2):275–283

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

Gélinas S, Desrochers M, Desrosiers J, Solomon MM (1995) A new branching strategy for time constrained routing problems with application to backhauling. Ann Oper Res 61:91–109

Hemmati A, Stålhane M, Hvattum LM, Andersson H (2015) An effective heuristic for solving a combined cargo and inventory routing problem in tramp shipping. Comput Oper Res 64:274–282

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

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon MM (eds) Column generation, chap 2. Springer, New York, pp 33–66

Norstad I, Fagerholt K, Hvattum LM, Arnulf HS, BjØrkli A (2015) Maritime fleet deployment with voyage separation requirements. Flex Serv Manuf J 27(2–3):180–199

Padberg M (1973) On the facial structure of set packing polyhedra. Math Program 5(1):199–215

Rasmussen MS, Justesen T, Dohn A, Larsen J (2012) The home care crew scheduling problem: preference-based visit clustering and temporal dependencies. Eur J Oper Res 219:598–610

Reinhardt LB, Clausen T, Pisinger D (2013) Synchronized dial-a-ride transportation of disabled passengers at airports. Eur J Oper Res 225(1):106–117

Ronen D (1983) Cargo ships routing and scheduling: survey of models and problems. Eur J Oper Res 12(2):119–126

Ronen D (1993) Ship scheduling: the last decade. Eur J Oper Res 71(3):325–333

Ryan DM, Foster B (1981) An integer programming approach to scheduling. Computer scheduling of public transport. Urban passenger vehicle and crew scheduling. In: Proceedings of an international workshop, pp 269–280

Sigurd MM, Ulstein NL, Nygreen B, Ryan DM (2005) Ship scheduling with recurring visits and visit separation requirements. In: Desaulniers G, Desrosiers J, Solomon MM (eds) Column generation. Springer, New York, pp 225–245

Steinzen I, Suhl L, Kliewer N (2009) Branching strategies to improve regularity of crew schedules in ex-urban public transit. OR Spectrum 31(4):727–743

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

Stålhane M, Andersson H, Christiansen M (2015) A branch-and-price method for a ship routing and scheduling problem with cargo coupling and synchronization constraints. EURO J Transp Logist 4(4):421–443

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

Vilhelmsen C, Larsen J, Lusby RM (2015) Tramp ship routing and scheduling—models, methods and opportunities. Technical Report, Department of Management Engineering, Technical University of Denmark

Vilhelmsen C (2014) *Tramp ship routing and scheduling—incorporating additional complexities*. Ph.D. thesis, Technical University of Denmark. Available via *findit.dtu.dk*