Tramp Ship Routing and Scheduling - Incorporating Additional Complexities

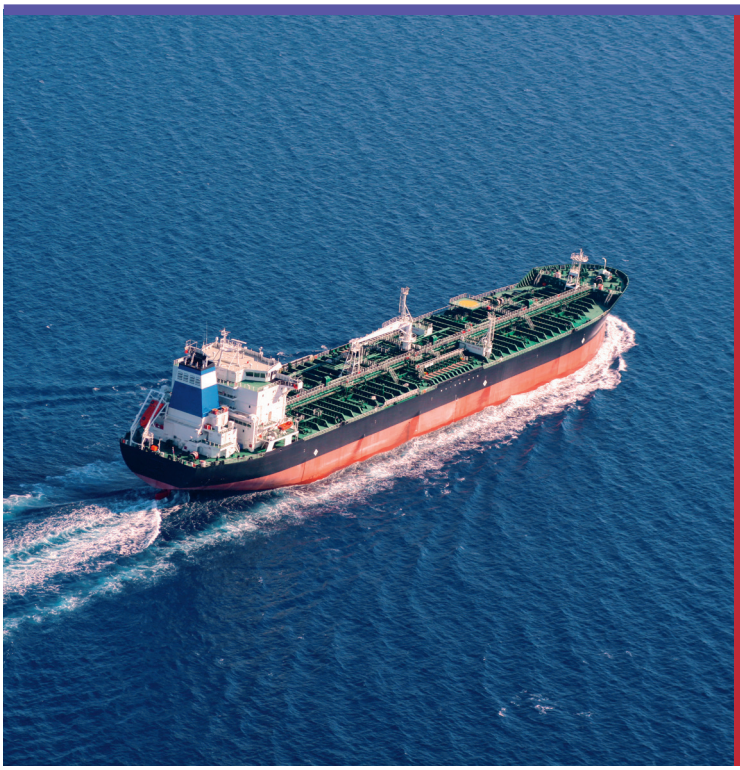
Vilhelmsen, Charlotte; Larsen, Jesper; Lusby, Richard Martin

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Tramp Ship Routing and Scheduling

– Incorporating Additional Complexities



PhD thesis 1.2015

DTU Management Engineering

Charlotte Vilhelmsen
August 2014

DTU Management Engineering
Department of Management Engineering

PhD thesis

Tramp Ship Routing and Scheduling - Incorporating Additional Complexities

Charlotte Vilhelmsen

August 3, 2014

Dansk titel:
Ruteplanlægning i trampfart - inddragelse af yderligere kompleksiteter
Type: Ph.d.-afhandling

Forfatter: Charlotte Vilhelmsen

ISBN:

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet, Building 426,
DK-2800 Kgs. Lyngby, Denmark
Phone: +45 45 25 48 00, Fax: +45 45 25 48 05
phd@man.dtu.dk

August, 2014

Summary

In tramp shipping, ships operate much like taxies, following the available demand. This contrasts liner shipping where vessels operate more like busses on a fixed route network according to a published timetable. Tramp operators can enter into long term contracts and thereby determine some of their demand in advance. However, the detailed requirements of these contract cargoes can be subject to ongoing changes, e.g. the destination port can be altered. For tramp operators, a main concern is therefore the efficient and continuous planning of routes and schedules for the individual ships. Due to mergers, pooling, and collaboration efforts between shipping companies, the fleet sizes have grown to a point where manual planning is no longer adequate in a market with tough competition and low freight rates.

This thesis therefore aims at developing new mathematical models and solution methods for tramp ship routing and scheduling problems. This is done in the context of Operations Research, a research field that has achieved great success within optimisation-based planning for vehicle routing problems and in many other areas.

The first part of this thesis contains a comprehensive introduction to tramp ship routing and scheduling. This includes modelling approaches, solution methods as well as an analysis of the current status and future direction of research within tramp ship routing and scheduling. We argue that rather than developing new solution methods for the basic routing and scheduling problem, focus should now be on extending this basic problem to include additional complexities and develop suitable solution methods for those extensions. Such extensions will enable more tramp operators to benefit from the solution methods while simultaneously creating new opportunities for operators already benefitting from existing methods.

The second part of this thesis therefore deals with three distinct ways of extending the basic tramp ship routing and scheduling problem to include additional complexities. First, we explore the integration of bunker planning, then we discuss a possible method for incorporating tank allocations and finally, we consider the inclusion of voyage separation requirements. For each of these extensions, we develop a new solution method and discuss the impact of incorporating these additional complexities.

Aside from a comprehensive introduction to tramp ship routing and scheduling, the main contribution of this thesis is the exploration of the three aforementioned extensions of the basic tramp ship routing and scheduling problem. The work on these three distinct extensions together represent a diverse collection of both problems and solution methods within tramp ship routing and scheduling.

Resumé (Summary in Danish)

Indenfor trampfart sejler skibene i stor stil som taxaer efter den tilgængelige efterspørgsel. Dette er forskelligt fra liniefarten, hvor skibene sejler mere som busser efter et på forhånd fastlagt rutenetværk og en offentliggjort tidsplan. Tramp operatører kan indgå længerevarende kontrakter og på den måde fastlægge en del af deres fremtidige efterspørgsel. Imidlertid kan dele af oplysningerne angående sådanne kontraktlaster løbende ændre sig. F.eks. kan destinationshavnen ændres. Effektiv og kontinuerlig ruteplanlægning for de individuelle skibe i flåden er derfor et vigtigt problem for tramp operatører. Sammenlægninger og samarbejder imellem rederier har fået flådestørrelserne til at vokse til et punkt, hvor manuel planlægning ikke længere er tilstrækkelig i et marked, hvor konkurrencen er hård og fragtpriserne lave.

Formålet med denne afhandling er derfor at udvikle nye matematiske modeller og løsningsmetoder til ruteplanlægning indenfor trampfart. Dette udføres indenfor rammerne af Operationsanalyse, et forskningsområde der har opnået stor succes med optimerings-baseret planlægning af rutelægning for biler og mange andre områder.

Første del af denne afhandling indeholder en grundig introduktion til ruteplanlægning indenfor trampfart. Det inkluderer tilgange til modellering, løsningsmetoder og desuden en analyse af den nuværende status og fremtidige retning for forskning indenfor ruteplanlægning i trampfart. Vi argumenterer for, at fremfor at udvikle nye løsningsmetoder til det basale rutelægningsproblem, så bør fokus nu ligge på at udvide det basale problem til at inkludere yderligere kompleksiteter. Sådanne udvidelser vil muliggøre, at flere tramp operatører kan gavne af løsningsmetoderne og samtidig skabe nye muligheder for operatører, der allerede har gavn af de eksisterende metoder.

Anden halvdel af denne afhandling omhandler derfor tre forskellige måder at udvide det basale trampfarts rutelægningsproblem til at inkludere yderligere kompleksiteter. Først undersøger vi integrationen af brændstofsplanlægning, så diskuterer vi en mulig metode til at inkorporere tank allokeringer og til sidst betragter vi inklusionen af krav til rejseadskillelser. For hver af disse udvidelser udvikler vi en ny løsningsmetode og diskuterer effekten af at inkorporere disse yderligere kompleksiteter.

Det største bidrag fra denne afhandling er, udover en grundig introduktion til ruteplanlægning indenfor trampfart, undersøgelsen af de tre førnævnte udvidelser af det basale ruteplanlægningsproblem. Arbejdet med disse tre forskellige udvidelser repræsenterer tilsammen en mangfoldig samling af både problemer og løsningsmetoder til ruteplanlægning indenfor trampfart.

Preface

This thesis has been submitted at Department of Management Engineering, Technical University of Denmark in partial fulfillment of the requirements for acquiring the Doctor of Philosophy degree.

The PhD study has been partly funded by the Danish Maritime Fund and conducted at Department of Management Engineering, Technical University of Denmark from January 2010 to August 2014. The work was supervised by Professor Jesper Larsen and co-supervised by Associate Professor Richard Lusby, both from Department of Management Engineering, Technical University of Denmark.

The thesis deals with different aspects of optimisation within tramp shipping and consists of two parts. The first part serves as an introduction to tramp shipping in general and to the specific topics explored during the PhD study. This part also contains a detailed overview of the work conducted during the PhD study as well as an overall conclusion on the conducted study. The second part of the thesis contains a collection of three research papers written during the PhD study. These research papers are all co-authored and self-contained with individual bibliographies.

Kgs. Lyngby, Denmark, August 2014

Charlotte Vilhelmsen

Acknowledgements

First, I want to thank my supervisor Professor Jesper Larsen for embarking on this journey with me into an industry neither of us knew much about when the project started. Your guidance, support and continuing optimism is what kept me going through these last years and there is no doubt that I could not have completed my journey without your help. Next, I want to thank my brilliant co-supervisor Associate Professor Richard Lusby for being my go-to-guy for everything from proofreading to implementation. Your advice and help along the way have been an invaluable contribution to this project.

I also want to thank my colleagues at DTU Management Engineering for many fruitful discussions and in general for providing a great work atmosphere. In particular I want to thank Line Blander Reinhardt for sharing her office with me, helping me with practical matters and most of all for being a fantastic friend through these last years. Berit Dangaard Brouer also deserves a special thanks for her help and support on both personal and work related matters.

Without any prior knowledge of the shipping industry, this project has at times had me going in circles. Therefore, I feel very privileged to have worked with Professor Kjetil Fagerholt at NTNU in Trondheim. His knowledge of and experience within optimisation of maritime transportation seem endless, and his willingness to share this with me has provided invaluable guidance for this project. I want to thank him for his support and for letting me visit him and his colleagues at NTNU on several occasions. From their research group at NTNU, a special thanks goes to Inge Norstad for helping me with data and advice for the project on evenly spread voyages.

This PhD project would never have seen the light of day if not for the financial support from The Danish Maritime Fund who partly funded the project. I thank them for this opportunity and further want to thank their administrator Carsten Melchiors for supporting me along the way and for forgiving me when the administrative side of the project was neglected.

I owe a great deal of my current knowledge about the tramp shipping industry to the experienced staff at Maersk Tankers A/S and I want to thank them for this. In particular I would like to thank Jakob Tørring for his help and advice on the bunker project.

Last, but certainly not least, I want to extend a heartfelt thanks to my family and friends for supporting me through these last years. In particular my dad has never failed to encourage me in this direction or to let me know how proud he is of me. Even as a small child, I remember him challenging me with small mathematical puzzles and supporting me when I dreamed of 'sitting in an office doing something math related' as opposed to becoming a nurse, firefighter or simply superwoman.

No doubt my children, my wonderful boys Alexander and Storm, have benefitted from my flexible working hours but suffered from my travels and periods of total work occupation. I love them with all my heart and thank them for their smiles, hugs and kisses and for providing me with an Operations Research free environment at home during this project. The closest thing we ever came to a work related discussion was when Alexander at 3 years old over the dinner table asked "so, mom, how did you decide the ships should sail today". Their "ignorance" and never ending love has seen me through the hard times of this project.

At the very bottom of this long list of acknowledgements I find the man who is at the very top of my list of people I love: My beloved husband Michael. Without your ongoing love, support, strength and confidence, I would have never embarked on this journey and certainly not completed it. You truly are my very best friend, the one I laugh with, the one I dream about, the one I live for, the one I love. Thank you for being mine.

Contents

Part I

Introduction and Theory

Chapter 1

Introduction

In 2012, global seaborne trade was estimated to have reached over 9 billion tons for the first time ever (UNCTAD, 2013). This translates into well over a tonne of cargo for every single individual on the planet, every single year, or equivalently 80% of world trade by volume. World trade therefore depends on the international shipping industry's efficiency and competitive freight rates. Furthermore, even though international shipping is the most carbon efficient mode of transport, the CO_2 emissions from the industry as a whole are still estimated to be around 3% of current global emissions and are, if nothing changes, expected to rise considerably as world trade increases (Michelsen, 2012). For several years now, the maritime sector has been subject to low freight rates caused by surplus fleet capacity and the weak economy. The combination of low freight rates and high bunker oil prices has made it difficult for many operators to produce earnings sufficient to cover just their minimum operating costs. The shipping industry itself, therefore, has as much incentive as any to improve their cost-effectiveness and recent years have shown increased exploration of strategies in this direction, e.g. slow steaming, which refers to the practice of operating ships at reduced speeds in order to save fuel. At the same time, the maritime sector also has an incentive to take their CO_2 emissions into consideration, both due to political pressure and to the fact that the industry itself is bound to be affected by the impacts of climate change, such as rising sea levels and more extreme weather. From all sides there is therefore an interest in research to increase efficiency within maritime transportation.

Within commercial shipping it is common to distinguish between three different basic operating modes although they need not be mutually exclusive: *liner*, *tramp* and *industrial* (Lawrence, 1972). Within liner shipping, which is primarily characterised by container shipping, vessels operate much like busses on a fixed route network according to a published timetable. In contrast, tramp ships operate much more like taxies following the available cargoes. Many tramp operators do however know some of the demand in advance as they can enter into long term agreements called *contracts of affreightment* (COAs). Such contracts state that the tramp operator is obliged to transport specified quantities of cargo between specified ports at a given rate during a specified time period. In addition to these contract cargoes, a tramp operator then tries to maximise profit from optional cargoes called *spot cargoes*. In industrial shipping, the ship operator is also the cargo owner and the objective is therefore to carry all the predefined cargoes at the minimum cost. Tramp and industrial shipping are primarily characterised by tankers and dry bulk carriers.

UNCTAD (2013) reports that although it is estimated that more than half of seaborne trade in dollar terms is containerized, this segment only accounts for 16% of global seaborne trade by volume. Furthermore, container ships only account for 13% of the world fleet deadweight tonnage while bulk carriers and oil tankers are responsible for respectively 42% and 30%. Similarly, UNCTAD (2013) reports that in 2012, bulk commodities accounted for nearly three quarters of the total ton-miles performed that year. Tramp and industrial operators are therefore accountable for a massive part of the global fleet as well as the total ton-miles performed each year, and hence even small improvements in efficiency within these operating modes can be expected to have great impact both from an economic and an environmental aspect.

As such, tramp shipping is not characterised by large economies of scale. Therefore, this shipping mode is generally not difficult to enter and has previously been comprised of many small

operators. Perhaps this is the reason why research within tramp shipping has previously lagged far behind that of industrial shipping (Christiansen et al., 2004). However, recent trends of mergers, pooling, and collaboration efforts between shipping companies have increased fleet sizes to a point where manual planning is no longer adequate (Christiansen et al., 2004). A further motivation for research within tramp shipping is that many companies previously involved in industrial shipping have now outsourced their transportation to independent shipping companies while some have even chosen to branch out and become more involved in the spot market in order to better utilise their existing fleet (Christiansen et al., 2004). Both these situations have created a shift from industrial to tramp shipping which, in combination with the increased need to minimise manual planning, has been reflected in the growing literature on research within tramp shipping and also motivated the tramp ship focus of this thesis.

Within tramp shipping, and for that matter also industrial shipping, the main focus in tactical planning, and to some extent also operational planning, is routing and scheduling the existing fleet. This thesis deals with tactical routing and scheduling where high-level routes and schedules are constructed. More detailed plans are made at the operational level once berth availability, weather conditions etc. are known. For tramp operators the tactical routing and scheduling problem boils down to determining which spot cargoes to transport, assigning all contract cargoes as well as chosen spot cargoes to specific ships while simultaneously finding the sequence and timing of port calls for all ships. The need to decide which spot cargoes to transport can greatly affect the requirements for solution methods. This is for instance the case if a broker calls the tramp operator with a specific cargo request and wants to know almost immediately whether or not the tramp operator is able and willing to transport the given cargo. This need to provide a yes/no answer very quickly means that solution methods with short running time are required. This is in contrast to most other tactical planning problems where running time is rarely an issue.

The tough competition between operators in todays low market adds pressure to devise the most efficient fleet schedules to properly utilise the existing fleet. However, as already mentioned, recent years have shown an increase in fleet sizes to a point where the construction of efficient schedules is both very time consuming and extremely difficult even for the most experienced planner. As an example we note that one of the collaborating partners during this PhD study, the pool manager Handytankers, is responsible for around 100 product tankers transporting petroleum products worldwide. At the same time, uncertainty plays a big part in maritime optimization where planners face a constantly changing environment with large daily variations in demand and many unforeseen events. Therefore, it is often necessary to re-plan routes and schedules continuously to accommodate new cargoes and changes to existing plans. This is even more relevant for tramp operators than for industrial operators due to the interaction with the spot market.

Hence, there is a need for an automated approach to this dynamic and ongoing planning problem that can both aid the construction of efficient schedules and enable fast changes to existing schedules in case of new or changed customer demands. Some commercial tools for optimisation of vessel fleet scheduling has been developed. Even so, many tramp operators still use experienced planners to manually route and schedule their fleets. We reflect on some of the reasons for this later in this thesis; however, for now we note that further work is needed in this area. The main goal of this PhD study has therefore been the development of new mathematical models and solution methods for tramp ship routing and scheduling.

Note that from an operations research point of view, the industrial ship routing and scheduling problem (ISRSP) can be viewed as a special case of the tramp ship routing and scheduling problem (TSRSP) where there are no spot cargoes. In such a case, the profit maximising tramp objective can be replaced by a cost minimisation objective as used in industrial shipping. Furthermore, compared to the fixed number of cargoes to carry in industrial shipping, the addition of spot cargoes in tramp shipping also yields greater flexibility suggesting greater financial impact from schedule optimisation within TSRSP than within ISRSP. The fact that TSRSP can be viewed as a generalisation of ISRSP, and that the financial impact from TSRSP can be expected to be greater than that of ISRSP, yields further motivation for focusing on tramp shipping as opposed to industrial shipping.

1.1 Thesis Structure

This thesis is divided into two separate parts. The first part contains an introduction to tramp shipping and the specific topics explored during this PhD study. Aside from this introductory chapter, Part I consists of five separate chapters:

- In Chapter 2 we discuss in further details the main characteristics of the basic TSRSP and also provide an example of a mathematical formulation for this problem. We also relate the TSRSP to vehicle routing problems and discuss some distinctions between these two related problems. Finally, we review literature on the TSRSP as well as commonly used solution methods for the this problem.
- In Chapter 3 we review the current status of research and applications within TSRSP and use this to reflect on the possible future research directions. We argue that, rather than developing new solution methods for the basic TSRSP, the main research focus should now be to extend this basic problem to include additional complexities. We use this argument to further motivate the specific research focus of this PhD study and give a brief overview of the topics explored in Part II of this thesis. Finally, we review literature on such extensions of the basic TSRSP.
- Chapter 4 contains a description of the common methods and tools used in Part II of this thesis and also relates these methods and tools to literature within tramp shipping.
- In Chapter 5 we list the main contributions of the work conducted during this PhD study. This includes a detailed overview of the work contained in the research papers that constitute Part II of this thesis.
- Finally, in Chapter 6 we arrive at a conclusion on the work of this thesis. We also highlight the contributions from this thesis and reflect on the directions for future research within tramp ship routing and scheduling.

The second part of this thesis contains a collection of three self-contained research papers written during the PhD study:

- In Chapter 7 we describe a new extension of the basic TSRSP to also include bunker planning. This work contains a detailed description of this new problem as well as a mathematical formulation for it. We also devise a solution method for it and provide a computational study on the effect of incorporating bunker planning into the routing and scheduling phase.
- In Chapter 8 we explore the Tank Allocation Problem and develop a randomised heuristic for finding feasible solutions to this problem. We also modify an earlier optimality-based method and create a hybrid method by combining our heuristic with this modified optimality-based method. This hybrid method is sufficiently fast to be used as a subproblem solution method in an extension of the basic TSRSP that includes the tank allocation aspect.
- In Chapter 9 we consider a previously studied extension of the TSRSP which includes requirements on a minimum time spread between some consecutive voyages. We present a new, exact method for solving this problem and provide a computational study to explore the efficiency of this method compared to a previously used method.

Two of the research papers have been submitted for publication in international journals, and one of these has been accepted for publication at the time of writing. The last paper will also be submitted and is simply awaiting approval from collaborating partners. We note that minor changes have been made to the layout of these papers to give the thesis a uniform look. However, the content of each paper is exactly the same as the content submitted to the journals.

Chapter 2

The Tramp Ship Routing and Scheduling Problem

In this chapter we use Section 2.1 to describe the main characteristics of the TRSRP and also discuss some of the operator specific characteristics of the problem. In Section 2.2 we present a mathematical model for a specific version of the TSRSP, while we use Section 2.3 to reflect on the similarities and distinctions between the TSRSP and vehicle routing problems. In Section 2.4 we review literature for the TSRSP, and, finally, in Section 2.5 we discuss solution approaches for the TSRSP.

2.1 Problem Description

The fleet size and mix is determined at the strategic planning level. In the TRSRP we therefore assume a fixed heterogeneous fleet comprised of ships of different sizes, load capacities, bunker consumptions, speeds, and other characteristics. Since ships operate around the clock, some ships can be occupied with prior tasks when planning starts, so each ship is further characterized by the time it is available for service and the location at which it is when it becomes available. The characteristics of each ship determine which cargoes, ports and canals it is compatible with, e.g. the draft of a ship can prohibit it from entering a shallow port and thereby make the ship incompatible with all cargoes being either loaded or discharged in this specific port. For some ships there can also be maintenance requirements during the planning horizon, and these must be respected in the scheduling process.

During the planning horizon, the tramp operator is obliged to transport a given list of contracted cargoes and can then turn to the spot market to derive additional revenue from spot cargoes if fleet capacity allows it, and if it is profitable. Each cargo is mainly characterized by the quantity to be transported, the revenue obtained from transporting it, and the pickup and discharge port. There is also a ship specific service time in port for loading and discharging, and a time window giving the earliest and latest start for loading. In some cases there is also a time window for discharge.

Since we consider a fixed fleet, we can disregard the fixed setup costs and focus on the variable operating costs which consist mainly of ship dependent fuel and port costs. Other costs can be relevant depending on the specific operator.

The objective of the TSRSP is then to create a profit maximizing set of fleet schedules, one for each ship in the fleet, where a schedule is a sequence and timing of port calls representing cargo loading and discharging. The optimal solution therefore combines interdependent decisions on which optional cargoes to carry, the assignment of cargoes to ships, and the optimal sequence and timing of port calls for each ship.

The above problem description is generic, and further operator specific characteristics are needed to properly model and solve the problem. Below we state some key operator characteristics and refer the reader to Christiansen et al. (2007) for further details on each of these, including mathematical arc flow formulations and some related literature.

- **Full shiploads or multiple cargoes:** Full shiploads correspond to the case where a ship can at most carry one cargo at a time, while multiple cargoes corresponds to the case where each ship can carry multiple cargoes at once. The full shipload case yields a much simpler mathematical formulation since the tasks of picking up and discharging each cargo can be aggregated into one task since no other cargoes can be handled in between. This also means that capacity constraints as well as precedence and coupling constraints are implicitly handled in the preprocessing phase.

- **Fixed or flexible cargo sizes:** Quite naturally, the fixed cargo size case refers to the case where each cargo size is a fixed quantity whereas the flexible case refers to the situation where each cargo size is given in an interval. Cargo revenues are, in the flexible case, defined as an amount per unit transported. Note that in the full shipload case, the same model can be used for both fixed and flexible cargo quantities since the specific cargo quantity transported by each ship can be determined from ship capacity during preprocessing along with the corresponding revenue.

- **Mixable or multiple (non-mixable) products:** In the case of mixable products, different cargoes are compatible with each other whether they consist of the same product or of multiple mixable ones. In either case, different cargoes can be loaded onboard a ship with no consideration to the type of products already onboard. In the case of multiple (non-mixable) products, different cargoes are not necessarily compatible and must be transported in different tanks onboard the ship in order to be onboard simultaneously. This situation leads to even further problem characteristics as the tanks can be of either fixed or flexible sizes.

- **Disallowing or allowing spot vessels:** If the capacity of the fixed fleet is not sufficient to carry the contracted cargoes, it can be allowed to charter in outside spot vessels to transport these cargoes at a specified cost. There can even be cases where fleet capacity is actually sufficient, but where it is simply profitable to charter in outside vessels to transport some contract cargoes thereby freeing up fleet capacity to instead transport spot cargoes.

A typical example of the full shipload case is the transportation of crude oil. Multiple cargoes, on the other hand, is common within dry bulk shipping and also for transportation of chemicals and refined oil products. Although the pool manager Handytankers mentioned in Chapter 1 transports refined oil products, they mostly sail full shiploads. Even though they have a fleet size of around 100 ships, this means that their planning problem can easily be less complex than that of an operator with "only" 50 ships carrying multiple cargoes. Therefore, the fleet size alone does not give an accurate indication of the complexity of the corresponding planning problem. However, in Chapter 1 we had not introduced the above problem characteristics; hence, the fleet size was used as a crude indication of problem complexity.

Flexible cargo sizes are often seen within transportation of bulk products. In particular, transportation of liquid products more or less implies flexible cargo sizes in order to prevent sloshing in partially empty tanks during sailing, and also to ensure stability of the ship. Within transportation of liquid products we also find transportation of chemicals, and this segment is a typical example of the multiple product case. A chemical tanker can have as many as 50 different tanks and hazardous materials regulations play a major role when allocating the products to the different tanks. E.g. products in neighboring tanks must be non-reactive and incompatible products must not succeed each other in a tank unless it is cleaned.

As far as we know, the use of spot vessels is not typical for any specific shipping segment but is a much more operator specific problem characteristic.

2.2 Mathematical Formulation

The research presented in this thesis covers a broad range of problem types using different combinations of the above operator specific characteristics. As can be seen in Christiansen et al. (2007), the different combinations lead to associated mathematical formulations differing in both size and complexity. It is not the aim of this section to present all the different mathematical formulations

derived from these main characteristics. Neither is it the aim to repeat the specific mathematical formulations used in the scientific papers written as part of this PhD study and presented in Part II of this thesis. However, it is the aim to provide the reader with a basic understanding of the structure of the TSRSP. Therefore, we here present an arc flow formulation for a simple version of the TSRSP, namely the one with full shiploads and without spot vessels. Note that due to the assumption of full shiploads, it does not really matter whether we assume fixed or flexible cargo sizes, or if we assume mixable or multiple products.

So, let $\mathcal{V}$ be the set of ships, and let $\mathcal{N}$ denote the set of cargoes to be transported during the planning horizon. We partition the cargo set $\mathcal{N}$ into the two smaller and disjoint sets, $\mathcal{N}_C$ and $\mathcal{N}_O$, containing, respectively, the contracted cargoes and the optional spot cargoes.

In order to define the problem on ship specific graphs, we define an origin node and a destination node for each ship $v \in \mathcal{V}$ and denote these $o(v)$ and $d(v)$ respectively. The origin node corresponds to the geographical location of the ship when planning starts, while the destination node is artificial and simply corresponds to the geographical location of ship v at the end of the planning horizon. We can also represent each cargo $i \in \mathcal{N}$ as a network node, and this node corresponds to the full transportation of cargo i, i.e. to both the pickup port and the discharge port of cargo i. Due to port and cargo compatibility, capacity and time requirements, as well as other restrictions, not all ships can transport all cargoes. Therefore, we define $\mathcal{N}^v$ as the set of nodes that ship v can visit, i.e. the nodes $o(v)$ and $d(v)$ as well as all nodes corresponding to cargoes, that can be transported by ship v. We further define ship specific arc sets, $\mathcal{A}^v$, containing all arcs $(i,j) \in \{\mathcal{N}^v \times \mathcal{N}^v\}$ where it is possible for ship v to transport cargo j directly after transporting cargo i. Note that $\mathcal{A}^v$ also contains the arc $(o(v), d(v))$ corresponding to ship v being idle during the entire planning horizon.

For each cargo $i \in \mathcal{N}^v$ we have a time window $[a_i^v, b_i^v]$ describing the earliest and latest time to start service for this cargo, when transported by ship v. For $o(v)$ this window is collapsed into the time, ship v is available for service. For any arc $(i,j) \in \mathcal{A}^v$, T_{ij}^v denotes the fixed time from arrival at the pickup port of cargo i to the arrival at the pickup port of cargo j and includes any port time for pickup and discharge of cargo i. For each arc $(i,j) \in \mathcal{A}^v$ we also have a ship specific profit, P_{ij}^v, which takes into account the revenue incurred from transportation of cargo i, the ship dependent port costs from transportation of cargo i, the sailing cost of transporting cargo i on ship v, and finally the cost of traveling ballast with ship v from the discharge port of cargo i to the pickup port of cargo j.

For the mathematical formulation we define binary flow variables x_{ij}^v for $v \in \mathcal{V}$, $(i,j) \in \mathcal{A}^v$ that are equal to 1, if ship v transports cargo i just before transporting cargo j, and 0 otherwise. The start time for service for each cargo is also variable; hence we define time variables t_i^v for each $v \in \mathcal{V}$ and $i \in \mathcal{N}^v$.

We can now give an arc flow formulation of this simple version of the TSRSP:

$$\max \sum_{v \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}^v} P_{ij}^v x_{ij}^v \tag{2.1}$$

s.t.

$$\sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}^v} x_{ij}^v = 1, \qquad \forall i \in \mathcal{N}_C, \tag{2.2}$$

$$\sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}^v} x_{ij}^v \leq 1, \qquad \forall i \in \mathcal{N}_O, \tag{2.3}$$

$$\sum_{i \in \mathcal{N}^v} x_{o(v)i}^v = 1, \qquad \forall v \in \mathcal{V}, \tag{2.4}$$

$$\sum_{j \in \mathcal{N}^v} x_{ji}^v - \sum_{j \in \mathcal{N}^v} x_{ij}^v = 0, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{N}^v \setminus \{o(v), d(v)\}, \tag{2.5}$$

$$\sum_{i \in \mathcal{N}^v} x_{id(v)}^v = 1, \qquad \forall v \in \mathcal{V}, \tag{2.6}$$

$$x_{ij}^v (t_i^v + T_{ij}^v - t_j^v) \leq 0, \qquad \forall v \in \mathcal{V},\ (i,j) \in \mathcal{A}^v, \tag{2.7}$$

$$a_i^v \leq t_i^v \leq b_i^v, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}^v, \tag{2.8}$$

$$x_{ij}^v \in \{0,1\}, \qquad \forall v \in \mathcal{V},\ (i,j) \in \mathcal{A}^v. \tag{2.9}$$

The objective function (2.1) maximises the joint profit from all ships in the fleet. Constraints (2.2) and (2.3) ensure that all contract cargoes are transported by exactly one ship, and that all spot cargoes are transported by at most one ship. Constraints (2.4) and (2.6) together with the flow conservation constraints in (2.5) ensure that each ship is assigned a schedule starting at the origin node and ending at the destination node. Constraints (2.7) ensure that if ship v transports cargo i directly before cargo j, the time for start of service for cargo j cannot begin before service start time for cargo i plus port time for transportation of cargo i plus travel time for transporting cargo i plus ballast travel time from the discharge port of cargo i to the pickup port of cargo j with ship v. Since waiting time is allowed, the constraints have an inequality sign. In constraints (2.8), the service time for ship v for cargo i, t_i^v, is forced to be within its time window, thereby also ensuring that no ship can start its schedule before it is available for service. Note that if ship v does not transport cargo i, the time variable t_i^v has no effect and is in some sense artificial. Finally, the flow variables are restricted to be binary in (2.9).

We note that constraints (2.7) are nonlinear and should in fact be:

$$x_{ij}^v = 1 \qquad \Rightarrow \qquad t_i^v + T_{ij}^v \leq t_j^v \qquad \forall v \in \mathcal{V},\, (i,j) \in \mathcal{A}^v. \tag{2.10}$$

However, as long as we require the x_{ij}^v variables to be binary, we can easily linearise these constraints by following the approach presented in Desrosiers et al. (1995). This approach introduces a large constant M_{ij}^v for each ship $v \in \mathcal{V}$ and each arc $(i,j) \in \mathcal{A}^v$. This constant, M_{ij}^v, must be the largest value that $t_i^v + T_{ij}^v - t_j^v$ can take, i.e. $M_{ij}^v \geq b_i^v + T_{ij}^v - a_j^v$. The linearised constraints then become

$$t_i^v + T_{ij}^v - t_j^v \leq M_{ij}^v(1 - x_{ij}^v), \qquad \forall v \in \mathcal{V},\, (i,j) \in \mathcal{A}^v.$$

Similar linearisation can be performed for nonlinear equations relating to capacity constraints and other constraints relevant depending on the problem characteristics. Thereby, the arc flow formulation for many TSRSPs, though not all, can easily be linearised. However, this is only relevant if we want to solve the arc flow formulations by use of standard commercial optimisation software for mixed integer linear programming. However, for most real life sized instances, this approach will be too time consuming.

2.3 Ship Routing vs. Vehicle Routing

As can be detected from the problem description in Section 2.1 and seen from the mathematical formulation in Section 2.2, the TSRSP is closely related to the Vehicle Routing Problem (VRP) and its many variants for which we refer the reader to Toth and Vigo (2002). Looking through literature, it is also quite clear that modelling approaches as well as solution methods for these two problems share great similarities, and that researchers within each of the two transportation modes have benefited from research within the other transportation mode. Literature also shows that research on the TSRSP has lagged far behind that of the VRP, though recent years have seen a huge increase in the amount of literature on the TSRSP. Therefore, the exchange of research ideas have most likely not been equally balanced between the two research communities. However, we note that one of the most successful solution methods from the vehicle routing literature is based on decomposition and dynamic column generation, and the application of this method for a pickup and deliver problem with time windows was first studied by Appelgren (1969) for a TSRSP application.

Even though the TSRSP is closely related to the VRP and its many variants, there are however important differences that facilitate the development of industry specific solution methods. Ronen (1983, 2002) elaborate on the operational differences between ships and trucks while Christiansen et al. (2004) add to this list of differences as well as reflect on similarities and differences between ships and trains, and ships and aircrafts. Rather than repeating it here, we refer the reader to Ronen (1983, 2002) and Christiansen et al. (2004) for an extensive discussion on the differences between ship routing and scheduling problems, and those of other transportation modes. Below we list a few of these differences including one specific for the tramp shipping industry.

- **Continuous operation:** As opposed to trucks, ships operate around the clock. This means that ship schedules usually do not have periods of idleness to absorb any delays. It also means

that ships do not only have different starting positions but also different starting times, as some ships can be occupied with prior tasks when planning begins.

- **No common depot:** Even in multi-depot versions of the VRP, vehicles must return to their home depot, whereas ships do not have to return to their starting point.
- **Compatibility issues:** In shipping, compatibility issues can arise between ships and cargoes, ships and ports, and ships and canals due to equipment, capacity, draft, size etc. Even the flag of the ship can prevent it from entering ports in countries that have political or cultural issues with the nation corresponding to the flag of the ship. Even more complicating, the draft of the ship is affected by the current load of the ship, and since this affects its compatibility with both ports and canals, the compatibility between ship and cargoes, ports and canals can be affected by the current onboard cargo.
- **Optional cargoes:** Specific for the TSRSP, the distinction between contract cargoes and optional spot cargoes leads to a priority on cargoes not used in standard vehicle routing problems where all customers must be serviced at minimum cost. In contrast, the tramp objective is to maximize profit as in the less known Pickup and Deliver Selection Problem, see Schönberger et al. (2003).

2.4 Literature on the TSRSP

As mentioned in Chapter 1, the three different basic operating modes, liner, tramp, and industrial, are not mutually exclusive. Especially tramp and industrial shipping are very closely related, and to some extent, the TSRSP can be viewed as a generalisation of the ISRSP. Thereby, it can be difficult as well as meaningless to separate the literature on these two problems into two distinct groups. Rather, most work on the TSRSP can be considered relevant for research within the ISRSP, and similarly, most work on the ISRSP can be considered relevant for research within the TSRSP. With research in these two fields dating as far back as the 1950s, the total amount of literature on these topics is quite extensive. It is therefore not our aim to provide a comprehensive review of all literature relevant for the TSRSP. For this, we instead refer the reader to the four review papers Ronen (1983), Ronen (1993), Christiansen et al. (2004) and Christiansen et al. (2013), which, on a decade basis, have provided the research community with comprehensive reviews on the latest research on ship routing and scheduling within all operating modes. Here, we instead review the very recent work on the TSRSP, and give some pointers to both recent and earlier work that together cover the different combinations of operator specific characteristics listed in Section 2.1. Note that since the purpose of this review is to frame the work presented in this thesis, we do not include the research papers from Chapters 7-9 here.

Even though the latest review paper was published in 2013, several papers on the TSRSP have been published since then. In 2012, Stålhane et al. (2012) extend the basic problem to allow split loads, i.e. allow each cargo to be split among several ships. They present a new path flow formulation and devise a Branch-Price-and-Cut procedure to solve the problem. The optimal cargo quantities are determined by solving multi-dimensional knapsack problems within the subproblems. Their computational results show that their solution method outperforms existing methods for certain types of instances. Kang et al. (2012a) consider the interaction between ship routing and scheduling, and ship deployment, though in a context quite different from ours. They use a commercial solver to directly solve their formulation.

Norstad et al. (2013) also consider a problem that combines ship routing and scheduling with ship deployment, although in a context very similar to the TSRSP. They add voyage separation requirements to the problem in order to perform similar voyages fairly evenly spread in time to reduce inventory costs for the charterer. 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 path generation and a commercial solver for the final problem. Their computational results show that both formulations can be used to solve smaller instances, while the path flow formulation can also be used to solve problems of more realistic sizes. Results also show that the inclusion of voyage separation requirements yields a much better spread of voyages and at little profit reduction. We return to this paper several times in this thesis, as it relates closely

to the work done in Chapter 9. Fagerholt and Ronen (2013) note that most research within this area has focused on solving simplified versions of reality. They present and consolidate results for three practical problems that each extend these simplified problem versions by adding further complexities and opportunities. The first extension considers flexible cargo quantities, the second allows split cargoes, while the third includes sailing speed optimization. Their results show that by using advanced heuristics, these extensions can, despite the increased problem complexity, be solved to achieve significantly better solutions. We discuss these findings further in Section 3.2.

Stålhane et al. (2014) combine traditional tramp shipping with a vendor managed inventory service in an attempt to challenge the traditional contract of affreightment. They 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 Branch-and-Price with a priori path generation. Larger instances are solved using a heuristic version of path generation. Their computational results show that the heuristic can significantly speed up computation time compared to the exact method, and at only a small reduction in solution quality. Their computational results also 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. Cóccola et al. (2014) present a novel column generation approach in which the conventional dynamic programming route-generator is replaced by a continuous time MILP slave problem. Their computational results show that their method outperforms both a pure exact optimisation model and a heuristic solution method previously reported in the literature. Castillo-Villar et al. (2014) present a Variable Neighborhood Search based heuristic procedure for solving the TSRSP with discretised time windows. They ignore spot cargoes and therefore seek to minimise cost rather than to maximise profit. The discretisation approach allows them to incorporate several practical extensions and they specifically investigate the inclusion of variable speed. Their computational results show that the heuristic is able to find good solutions within reasonable computation time. It is, however, only tested on smaller instances. Finally, somewhere between tramp shipping and liner shipping, Moon et al. (2014) investigate a combined ship routing and fleet deployment problem. They propose a genetic algorithm with local search to solve the problem, as well as a simple heuristic. Their computational results show that the genetic algorithm with local search is most efficient.

We note that the first review paper (Ronen, 1983) contains only two references on tramp shipping, while the second (Ronen, 1993) does not contain any. The third review paper (Christiansen et al., 2004) lists five new tramp references, though some of these are for problems with a mix of tramp and industrial shipping. Finally, the fourth review paper (Christiansen et al., 2013) lists around 30 papers related to the TSRSP, though some of these are specifically for industrial shipping. These numbers show a clear trend of increased research interest within the TSRSP, and we note that we above listed eight references on the TSRSP from just within the last year.

Examples of TSRSPs for full shiploads can be found in Appelgren (1969, 1971) and Norstad et al. (2013), and we note that the work presented in Chapters 7 and 9 is also for full shiploads. The multiple cargo case can be found in e.g. Korsvik et al. (2010), Andersson et al. (2011b) and Cóccola et al. (2014). Although the work presented in Chapter 8 does not directly consider the TSRSP, it does however consider a subproblem for routing and scheduling tramp fleets carrying multiple cargoes.

Although flexible cargo sizes relates to most operators transporting bulk products, and in particular liquid products, this problem characteristic is neglected in most work on the TSRSP. We do however find a few examples on flexible cargoes, e.g. Brønmo et al. (2007b), Brønmo et al. (2010) and Korsvik and Fagerholt (2010). Also, we note that for full shipload cases, such as the work considered in Chapters 7 and 9, flexible cargo sizes can be implicitly included since the specific cargo quantity transported by each ship can be determined from ship capacity during preprocessing along with the corresponding revenue.

Similarly, the added complexity of the multiple (non-mixable) products is rarely considered in the literature. A few examples of work in this area can be found in Fagerholt and Christiansen (2000a), Kobayashi and Kubo (2010), Oh and Karimi (2008) and Neo et al. (2006), although the latter only considers one ship. The work presented in Chapter 8 also relates to the multiple (non-mixable) products case.

Finally, we note that the inclusion of spot vessels only results in minor changes to the arc flow formulation from Section 2.2, and as such does not complicate the solution procedure further.

Therefore, the inclusion of this characteristic does not on its own constitute an interesting research area. Rather, it is included in a wide variety of papers, depending on the real life application considered in those papers. We note that spot vessels are included in the work presented in Chapter 9, where their presence does, however, complicate the solution procedure.

2.5 Solution Approaches for the TSRSP

Before diving into the existing solution methods for the TSRSP, we briefly want to clarify some terms used in relation to solution methods throughout this thesis. First, an *exact solution approach* is one that guarantees an optimal solution to the considered problem. Many such approaches require long running times; hence, there is an obvious trade off between solution quality and computation time. Therefore, in many cases a solution method that sacrifices optimality for the sake of shorter running time is often more appropriate. Such methods are called *heuristics*. Heuristic solution approaches are often tailor made for the problem at hand, though some have been generalised to so called *metaheuristics*, which constitute a generic heuristic framework for solving a wide range of problems. Finally, we use one last class of heuristics called *optimisation-based methods*, which contains methods that are based on exact solution approaches but where some components have been made heuristic to save computation time.

2.5.1 Exact Solution Approaches for the TSRSP

Most work on the TSRSP contain a problem specific arc flow formulation or refer to similar work, that contains such a formulation. As mentioned in Section 2.2, most of these formulations can easily be linearised, and, in theory, solved by use of standard commercial optimisation software for mixed integer linear programming. An example of this approach can be found in Norstad et al. (2013). However, just as Norstad et al. (2013) conclude, for most real life sized instances, this approach will be too time consuming.

A standard approach is therefore to reformulate the original model to a path flow formulation in which columns constitute feasible ship routes or schedules. This approach allows the ship specific constraints to be handled in subproblems, which generate the columns. These columns can either be generated a priori or dynamically. We discuss both these approaches in detail as well as related literature in Chapter 4, where we describe the methods and tools used in the work from the research papers presented in Chapters 7-9.

2.5.2 Heuristic Solution Approaches for the TSRSP

Optimisation-based methods for the TSRSP mainly relates to heuristic modifications of the above mentioned exact approach of column generation. As we discuss this method in details in Chapter 4, a discussion on optimisation-based methods, as well as related literature, can also be found in that chapter.

Regarding more general heuristics and metaheuristics, we can find numerous examples of such work in the literature. Brønmo et al. (2007a) solve the multiple cargo TSRSP using a multi-start local search heuristic. They compare their heuristic approach with an a priori column generation approach, and computational results show that the heuristic returns optimal or near-optimal solutions within reasonable time for real-life instances. Korsvik et al. (2010) solve the same problem using a tabu search heuristic and compare their method to the multi-start local search heuristic from (Brønmo et al., 2007a). Results from this comparison show that the tabu search heuristic performs better than the multi-start local search heuristic, especially for large and tightly constrained problems. Also for the multiple cargo TSRSP, Malliappi et al. (2011) present a variable neighborhood search heuristic, and compare their method to both the tabu search heuristic and the multi-start local search method, though with their own implementation of these two methods. Computational results from modified benchmark instances from land transportation show that the variable neighborhood search on average performs best.

Although using a different model than the above references, Lin and Liu (2011) also solve the multiple cargo TSRSP. They develop a genetic algorithm and show that it outperforms solving their mathematical formulation directly by commercial software. Kang et al. (2012b) and Jung

et al. (2011) also develop a genetic algorithm, though this is for a quite special TSRSP involving transportation of cars.

Korsvik and Fagerholt (2010) propose a tabu search algorithm for solving the TSRSP with flexible cargo sizes. Their compuational results show that this method provides optimal or near-optimal solutions within reasonable time for real-life instances. For project shipping with cargo coupling constraints, a special real-life TSRSP, Fagerholt et al. (2011) also propose a tabu search algorithm. This algorithm is a modified version of the one from Korsvik et al. (2010). Again, computational results show that the tabu search heuristic yields optimal or near-optimal solutions within reasonable time.

Yet another different approach is considered in Korsvik et al. (2011), where they allow split loads. They present a large neighbourhood search heuristic, and their results show this heuristic is able to provide good solutions to real-life instances within reasonable time. Finally, Norstad et al. (2011) allow variable speed in the TSRSP and solve this problem using a multi-start local search based on the work presented in Brønmo et al. (2007a).

Chapter 3

Incorporating Additional Complexities

In this chapter we first review the current status of the TSRSP in Section 3.1, and afterwards discuss a direction for current and future research within the area in Section 3.2. In Section 3.3 we frame the topics covered in this thesis in the context of this research direction. Finally, in Section 3.4 we present some literature consistent with and relevant for this research direction.

3.1 Current Status of the TSRSP

As noted in Section 2.4, recent years have shown a dramatic increase in the amount of research not just within maritime transportation in general but also specifically within tramp ship routing and scheduling. Complemented by the simultaneous improvements in both software and hardware, this has made the development of both heuristic and exact methods reach a level of efficiency that renders most realistic sized instances of the TSRSP now solvable within reasonable time.

Even so, many tramp operators still use experienced planners to manually route and schedule the fleet, and it is only natural to reflect on the reasons for this. Below we list some of the reasons we believe are responsible for this remaining gap between research and implementation.

- **Conservative industry:** The shipping industry has a long and proud tradition of using planners with practical experience from a background in shipping rather than a more academic background. Therefore, most planners are very skeptical towards optimisation-based tools not to mention reluctant to facilitate development of systems that could eventually make the company less dependent on them. Fagerholt (2004) presents some experience from the development and implementation of a decision support system for vessel fleet scheduling, and also describes the struggle to convince the planners of the value of such a system. At the same time, the conservative nature of the industry has made many shipping companies unwilling to share data for research projects, and without real life data, the gap between research and implementation naturally remains.

- **Industry under pressure:** Previously the research and technology were not at an adequate level to facilitate implementation of optimisation-based tools, and now that they are adequate, the industry is under such pressure that it is hard to find resources for anything that is not immediately income generating. Certainly, it can be hard to justify the allocation of resources to a research project that might in the future turn out to be valuable, when the company is at the same time struggling to create revenues matching just their minimum operating costs.

- **Inability to plan ahead:** Uncertainty is known to play a big part in maritime optimisation where planners face a constantly changing environment with large daily variations in demand and many unforseen events caused, among other things, by changing weather conditions. This uncertainty coupled with the long voyages spanning several days and sometimes even weeks,

makes it hard to plan far ahead. The financial impact, not to mention loss of goodwill, from not adhering to agreed cargo transportation is simply too big to allow planning far ahead. For some operators the uncertainty is so high that they cannot plan even a single voyage ahead. Instead they simply wait for a ship to become idle and then assign it the currently best transportation request. This is for instance the case in transportation of refined oil products where the discharge port and date can be unknown right up until actual discharge. A loaded ship simply waits at sea until oil prices in ports reach a satisfactory level. This can make the combinatorial puzzle, of finding the best match of ships to cargoes, seem simple and make it even harder for conservative planners to see the potential in optimisation-based tools.

- **Dynamic nature of problem:** The fact, that the basic TSRSP can be solved, is naturally a requirement for the development of a planning tool in this area. However, the basic TSRSP is a static and simplified version of reality. As already mentioned, there is great uncertainty in maritime transportation; hence, it can be necessary to continuously solve a modified version of the problem. This means that the fleet information must continuously be updated as well as cargo information. Note that even things such as the changing tidal conditions can affect the solution to the problem and create cause for reoptimisation. Similarly, the planning tool must somehow be linked to information from the spot market and continuously update these data. Overall, this means that the optimisation component itself will in many cases be just a very small part of the full planning tool and hence, implementation will lag behind that of research within methods for solving the TSRSP.
- **Modelling issues:** Most mathematical models are simplified versions of reality, but in shipping this is even more so. Aside from operational considerations completely ignored in the modelling process, those considerations that have been included in the models have been greatly simplified. In shipping there are a lot of constraints that are simply so difficult, if not impossible, to accurately model that simplification is the only option. To give a few examples we note that time windows are in reality often soft since extensions can be negotiated at a certain cost. However, this is not always possible; hence, modelling time windows as soft is not accurate either. Similarly, the constraints describing ship-port compatibility is affected by both the current load of the ship and the tidal conditions, and these constraints can even in some cases be of a rather subjective nature since a risk-willing captain can agree to sail a ship into a shallow port even if the draft of the ship should not allow this.
- **Simplified problem:** Adding to the above, the basic TSRSP is a simplified version of reality. In most cases, the problem must be extended to incorporate operator specific complexities in order for the solutions to be applicable in real life. Conversely, planners must find ways to modify solutions from the simplified problem in order to take into account the additional practical aspects and complexities ignored in the simplified problem.

3.2 Current and Future Research Direction for the TSRSP

The first two items, and to some extent also the third, on the above list do not as such suggest further research within tramp ship routing and scheduling, but rather that researchers take on a more consultant like role in order to better sell their ideas to the industry. At the same time it is probably a matter of giving the shipping industry a little more time to get to a point where they are both interested in and able to accept optimisation-based tools. In fact, the shipping industry has started to change in this direction by gradually employing more planners with more academic background and less practical experience. The third item on the list also require changes from within the industry itself. Operators will be able to plan further ahead if they are given more flexibility to cope with the high uncertainty, e.g. send a different ship than originally agreed, arrive later or earlier than planned etc.

The fourth and fifth item on the list focus mainly on the implementation side. They advocate the development of decision support tools that allow much interaction between the user and the planning tool to facilitate the construction of several good solutions as opposed to just one solution that is, at least on paper, optimal. The development of decision support tools is certainly interesting

and relevant work. However, the aim of this PhD project is the development of new mathematical models and solution methods for tramp ship routing and scheduling. Therefore, the implementation side is left as promising work for others.

The last item on the above list does however advocate further research within tramp ship routing and scheduling in the direction of development of new mathematical models and solution methods. However, this research should focus on extensions of the basic problem described in Chapter 2. The ability to solve such extended TSRSPs will hopefully facilitate more real life implementations since the corresponding extended models will fit the reality of a broader range of tramp operators. Furthermore, even operators previously able to benefit from solution methods for the basic TSRSP can benefit from these extensions as they can create new opportunities for profit maximisation as well as increased customer satisfaction. To give an example, we note that the basic TSRSP assumes that each ship sails at fixed speed while determination of the actual speed along each voyage leg is left for operational planning. Recent years has, however, shown quite a bit of research where the variable speed aspect is incorporated into the basic TSRSP and, not surprisingly, results show that operators can increase profits from such an approach, see e.g. Norstad et al. (2011). Below is a quote from Fagerholt and Ronen (2013) supporting the general idea of further research within extensions of the basic TSRSP:

"*This demonstrates that it is much more important to model and solve the right problem, considering the opportunities that often arise in practical problems, than to strive for optimal solutions to simplified versions of the problem.*"

In general, the development of research within tramp ship routing and scheduling over recent years is also consistent with the notion of extending the basic TSRSP to include additional complexities. In Section 3.4 we present an overview of this literature covering a broad range of extensions of the basic TSRSP.

3.3 Thesis Focus

Summing up, some tramp operators rely on further research on extensions of the basic TSRSP to be able to benefit from sophisticated planning tools while others will simply experience further benefits from such extensions. Furthermore, solution methods and technology are now advanced enough to facilitate the incorporation of additional complexities, and the aim of this thesis is therefore to consider just such extensions. More specifically, in Part II, we present three distinct research projects covering each their extension. Chapter 5 gives a detailed description of each of these; hence, below we just present a quick overview of them.

- **Incorporating bunker planning:** The first project explores the opportunities for and effects of integrating the additional complexity of bunker planning, which is normally considered an operational planning problem.

- **Incorporating tank allocation:** The second project aims at developing a solution method to find feasible tank allocations very quickly in order to allow the method to be used as a subproblem routine in the TSRSP extended to include the additional complexity of tank allocations. Tank allocation is, just as bunker planning, normally considered an operational planning problem.

- **Incorporating voyage separations requirements:** The third project aims at developing an efficient solution method for the TSRSP with the practical aspect of voyage separation requirements incorporated to improve customer service. The additional complexity of such separation requirements is not dealt with in the basic TSRSP; accordingly, planners adhering to optimisation tools must either ignore these requirements at the risk of poor customer service or they must manually modify the solutions from the basic TSRSP.

3.4 Literature for TSRSP extensions

Naturally, there will be quite different perceptions of what constitutes the 'basic TSRSP' and what qualifies as 'additional complexities'. Here, we let the existing literature on tramp ship routing and scheduling guide our definition of the basic problem. We do this by first noting that, although the multiple cargo case is much more complex than the full shipload case, plenty of literature deal with multiple cargoes. We already mentioned several examples in Section 2.4, and adding to this we note that, among many others, Norstad et al. (2011) also manage to incorporate further complexities in the multiple cargo problem as they in this paper allow variable speed. Therefore, although the multiple cargo case does add complexity compared to the full shipload case, in our definition of the 'basic TSRSP' we allow for both full shiploads and multiple cargoes. However, as we noted in Section 2.4, flexible cargo sizes are neglected in most work on the TSRSP. This stands in contrast to the fact that flexible cargo sizes are relevant for most tramp operators transporting bulk products, and in particular liquid products. Therefore, we consider this problem characteristic as an 'additional complexity'. Similarly, we noted in Section 2.4 that the multiple (non-mixable) product case is rarely considered in the existing literature on the TSRSP; hence, we do not consider this problem characteristic as part of the basic TSRSP. Finally, we noted in Section 2.4 that the inclusion of spot vessels does not as such complicate the solution procedure, and that this problem characteristic is included in a wide variety of papers on the TSRSP. Accordingly, in our definition of the basic TSRSP we allow the use of spot vessels, though they are certainly not required to be used.

Any complicating problem characteristic not already mentioned or implicitly excluded by the mathematical formulation of the problem in Section 2.2 (e.g. variable speed), can, in our opinion, be considered an 'additional complexity'. Below we list some examples of such additional complexities as well as literature on these examples.

We have already mentioned both flexible cargo sizes and multiple (non-mixable) products as additional complexities and provided examples of work on these topics in Section 2.4. Again, we note that due to the full shipload assumption in Chapters 7 and 9, the flexible cargo size assumption can be implicitly included during preprocessing. The work in Chapter 8 concerns a subproblem in the full solution procedure for a TSRSP with multiple (non-mixable) products; hence the inclusion of flexible cargo sizes will depend on the full solution procedure.

An extension of the basic TSRSP, which has received a lot of attention in recent years, is the incorporation of variable speed. This recent interest in speed optimisation has been motivated by both steep bunker fuel price increases as well as an increased focus on the environmental impact of maritime transportation (and transportation in general). Examples of routing and scheduling combined with speed optimisation can be found in Norstad et al. (2011) and Gatica and Miranda (2011).

Another topic that has recently received increased attention due to the increase in fuel prices, is the aspect of bunkering, i.e. refueling. The main attention on this topic has been directed towards liner shipping, see e.g. Plum et al. (2014, forthcoming) who present an overview of formulations, solution methods as well as results on this topic. In the majority of work on bunkering, the problem is solved for a fixed route, and this assumption is certainly obvious within liner shipping. However, since tramp ships do not sail according to fixed route networks, we have explored the integration of bunker planning in the TSRSP in Chapter 7.

Recent years have also shown an increase in literature involving several parts of the supply chain. The majority of this work is related to industrial shipping, where the supply chain aspect fits naturally, since the industrial ship operator is also the cargo owner; hence, he/she is also responsible for any inventory management at the ends of the maritime transportation legs. Such situations leads to a specific class of problems called *maritime inventory routing problems*. We refer the reader to Christiansen et al. (2013) for a thorough introduction as well as literature review on such problems. The inventory aspect is less obvious within tramp shipping. Even so, we note that Stålhane et al. (2014), as already mentioned, combine tramp shipping with a vendor managed inventory service. In this work the tramp operator has both mandatory cargoes that must be transported, optional cargoes that can be transported, as well as inventory pairs that may be serviced a number of times to keep inventories within their limits. Also mentioned previously, Norstad et al. (2013) add voyage separation requirements to the routing and scheduling problem

in order to reduce inventory costs for the charterer, and we consider basically the same problem in Chapter 9.

Although we can find other examples of additional complexities added to the basic TSRSP, we end this section by one last example, namely the split load case, where each cargo can be split among several ships. We have already mentioned that Korsvik et al. (2011) and Stålhane et al. (2012) consider this problem, and note that split loads are also considered in Andersson et al. (2011a).

Chapter 4

Methods and Tools

In Section 2.5 we discussed solution methods found in the literature, and in this chapter we discuss some common methods and tools used to solve the TSRSP extensions explored in the three research papers in Part II of this thesis. We also relate these methods and tools to literature within the area.

4.1 Column Generation

As can be seen from both Section 2.4 and 2.5, the methods used to solve the TSRSP and its extensions are quite diverse, ranging from simple heuristics to exact methods. Naturally, the choice of method must depend on the complexity of the problem at hand as well as the requirements for solution quality and computation time. Even though literature contains such a broad range of both problems and methods, one method seems to receive much more attention than any of the others: Column generation. The popularity of this method is in line with research within many other fields. We note that both crew scheduling and vehicle routing problems are very often solved using this approach, and that the method has achieved great success within these areas and many others, see e.g. Butchers et al. (2001) and Toth and Vigo (2002). Furthermore, Desaulniers et al. (2005) devoted two whole chapters to column generation in maritime problems. Looking through the literature, it seems that this approach has become more popular within recent years where focus has been on TSRSP extensions. This is on one hand a little surprising since the added complexity contained in these extensions should indicate the use of heuristic methods to achieve reasonable running times. On the other hand, the increase in software and hardware has allowed more focus on exact methods, and one of the main advantages of the column generation approach is that it allows complicating constraints to be handled in ship specific subproblems. Furthermore, it should be noted that even though this is an exact method, it can easily be modified in a heuristic direction as we discuss in Section 4.2. No matter the heuristic or exact nature of the method, the frequent use of the column generation approach for a broad range of complex instances, not just within shipping but also within many other research areas, demonstrates the great advantage and flexibility of this method that allows easy incorporation of complicating constraints. In Part II of this thesis, we present two scientific papers which both utilise column generation, as well as a third one which considers a problem that can be used as a subproblem in a column generation routine.

4.1.1 Master Problem

Since most of the TSRSP constraints relate to a specific ship, it seems natural to decompose the problem and use column generation. Whether we use a priori column generation or dynamic/delayed column generation, the master problem of such a decomposition is the same. It is a path formulation containing only the constraints which couple the ships together and ensure that each ship follows exactly one path. In the basic TSRSP, the coupling constraints correspond to constraints (2.2) and (2.3), though these must be expressed by new path flow variables while a similar reformulation of the original objective function (2.1) must be performed.

To arrive at this new path formulation for the basic TSRSP discussed in Chapter 2 as well as many of its extensions, we let $\mathcal{R}^v$ denote the set of all feasible schedules for ship v. We denote the profit of a schedule by p_r^v for $r \in \mathcal{R}^v$ and define a binary schedule variable λ_r^v that is equal to 1 if ship v sails schedule r, and 0 otherwise. The profit p_r^v is calculated based on information from the underlying schedule, which holds all necessary information, i.e. the ship it is constructed for, the cargoes carried, and the timing of port calls during the schedule. Finally, we let the parameter a_{ir}^v be equal to 1 if ship v carries cargo i in schedule r, and 0 otherwise.

The master problem is now given by the following path flow reformulation of the original arc flow model:

$$\max \sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}^v} p_r^v \lambda_r^v \tag{4.1}$$

s.t.

$$\sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}^v} a_{ir}^v \lambda_r^v = 1, \qquad \forall i \in \mathcal{N}_C, \tag{4.2}$$

$$\sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}^v} a_{ir}^v \lambda_r^v \leq 1, \qquad \forall i \in \mathcal{N}_O, \tag{4.3}$$

$$\sum_{r \in \mathcal{R}^v} \lambda_r^v = 1, \qquad \forall v \in \mathcal{V}, \tag{4.4}$$

$$\lambda_r^v \in \{0,1\}, \qquad \forall v \in \mathcal{V},\ r \in \mathcal{R}^v. \tag{4.5}$$

The objective function (4.1) maximises profit from chosen schedules. Constraints (4.2) and (4.3) are the path flow reformulations of constraints (2.2) and (2.3), respectively. The convexity constraints (4.4) and binary restrictions on the schedule variables in (4.5) ensure that each ship is assigned exactly one schedule. If slack variables are added to constraints (4.3), we recognise the master problem as a Set Partitioning Problem in which the columns corresponds to feasible ship schedules.

Allowing Convex Schedule Combinations

For some applications it can be acceptable to use convex combinations of different schedules for the same ship as an actual schedule for this ship, as long as the schedules included in the convex combination correspond to the same geographical route. This means that the binary restrictions in (4.5) can be replaced by restrictions of the form

$$\{r : \lambda_r^v > 0\} \quad \text{correspond to equal geographical routes,} \qquad \forall v \in \mathcal{V}. \tag{4.6}$$

This approach is used in e.g. Christiansen and Nygreen (1998) for a ship routing problem with inventory constraints. We also allow such convex schedule combinations in the paper presented in Chapter 9. However, as we conclude below when discussing the integer property of the master problem, unless further complexities are added to the master problem, this relaxation of the binary restrictions has no effect. With the solution method used in Chapter 9, this is exactly the case. Furthermore, there can be many situations where convex schedule combinations do not qualify as actual schedules, primarily due to cost calculations. One example is when the TSRSP is extended to allow variable speed as in Norstad et al. (2011). Since the fuel consumption function is a convex function of speed, once we include variable speed we can no longer allow convex schedule combinations, since the convex combination of schedule costs would overestimate the actual cost of the schedule. Therefore, we must select exactly one schedule for each ship. Another example is in the work presented in Chaper 7 where we extend the TSRSP to include bunker planning. In this case the schedules also contain information on bunker stops, and each of these stops corresponds to a predefined price for a specific bunker option at a given time. Assume now that two schedules for the same ship each contain a stop at a bunker option, B, though at different times denoted t_1^B and t_2^B with different bunker prices, P_1^B and P_2^B, respectively. The convex combination of these two schedules will have an arrival time somewhere in the interval (t_1^B, t_2^B), assuming that $t_1^B \leq t_2^B$. Due to price fluctuations, there is no guarantee that the bunker price in this interval corresponds to the convex combination of the bunker prices P_1^B and P_2^B. Therefore, in this situation as well, we must select one single schedule for each ship and so constraints (4.6) cannot be used.

Master Problem Integer Property

Before going into details with the actual column generation procedure, we want to briefly discuss the underlying structure of the master problem and the implied integer properties from this structure. To ease notation, we let V denote the number of vessels, $|\mathcal{V}|$, and by A^v we denote the $|\mathcal{N}| \times |\mathcal{R}^v|$ submatrix containing the schedules for ship $v \in \mathcal{V}$, i.e. the a^v_{ir} entries. If we insert slack variables, $s_1, \ldots, s_{|\mathcal{N}_O|}$, in constraints (4.3), then the constraint matrix looks as in Figure 4.1, where zero entries are ignored. The convexity constraints (4.4) are generalised upper bound constraints, and

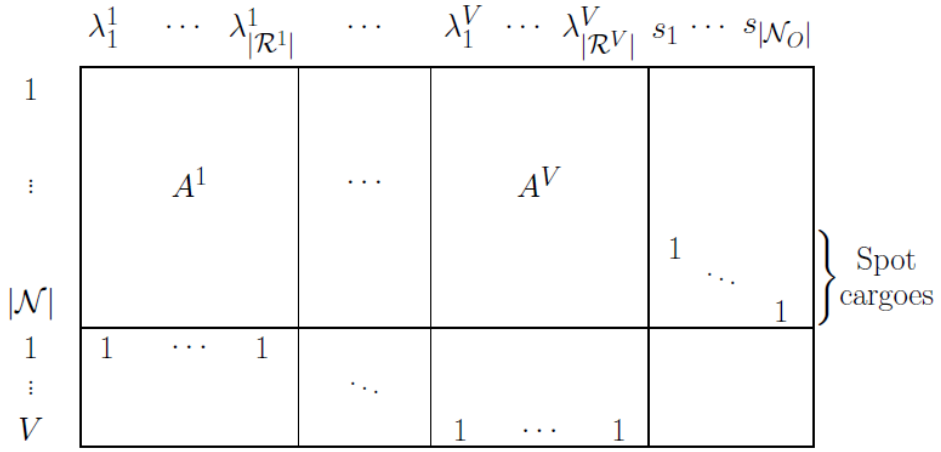


Figure 4.1: The master problem constraint matrix

because of these, the submatrix for each ship is perfect. This means that fractional solutions can never occur solely within one of the individual ship submatrices. Fractional solutions can, however, appear across submatrices for different ships. Thereby, the LP solution can only be fractional if two or more ships are competing for the same cargo. We refer the reader to Padberg (1973) and Conforti et al. (2001) for a discussion on perfect matrices and their properties.

Note that for problems with this integer property, once we have eliminated fractionalities across submatrices for different ships, the solution will never contain more than one schedule for each ship; accordingly, there is no point in checking for "equal geographical routes" as in constraints (4.6).

4.1.2 A Priori Column Generation

For the TSRSP, and most of its extensions, there is no need to include all feasible schedules for each ship in the master problem. Instead we can limit the column set to contain the profit maximising schedule for each combination of ship and cargo set. Due to the uncertainty involved in maritime transport as well as the long voyages, many operators are prohibited from planning more than a few voyages ahead for each ship. This is especially true for the full shipload case, and for such operators, depending on the constraints for ship-cargo compatibility, the number of feasible combinations of ships to cargo sets can be small enough to allow a priori full enumeration of all these combinations, and hence of all the columns in the master problem. Examples of tramp related work using the a priori column generation approach can be found in Kim and Lee (1997), Brønmo et al. (2007b), and Andersson et al. (2011b).

Generating the Columns

For each feasible cargo set for each ship, we therefore need to determine the profit maximising schedule, i.e the order and timing of port visits, that yields the best profit for this cargo set. Depending on the TSRSP considered, it can be necessary to determine other schedule details simultaneously, e.g. the load quantities in case of flexible cargo sizes. Naturally, only feasible schedules can be considered; hence, the timing of port visits must adhere to the cargo time windows, ship capacity must be respected and so on. Generally speaking, all constraints from the original

arc flow formulation that corresponds to the considered ship, must be adhered to in this column generation process.

The column generation in the a priori approach therefore consists of repeatedly determining the optimal route and schedule for a given ship and a given cargo set. This can be done by simply enumerating all feasible routes for the given cargo set or by dynamic programming. The enumeration approach is used in e.g. Brønmo et al. (2007b). The dynamic programming approach, on the other hand, is used in e.g. Fagerholt and Christiansen (2000b).

4.1.3 Dynamic Column Generation

Some instances of the TSRSP and its extensions are too large to allow full enumeration of all combinations of ships to cargo sets. There can also be situations where full enumeration is actually possible, but where additional coupling constraints in the master problem means that the profit maximising schedule for one ship and cargo set is not necessarily profit maximising when taking the entire fleet into account. This is the case in the scientific paper presented in Chapter 9. There, we have temporal dependencies between the schedules for different ships, and these dependencies require us to enumerate all feasible schedules rather than just all profit maximising schedules. Therefore, for several reasons, there can be situations where we are prohibited from using the a priori column generation approach. In such situations, we use dynamic column generation, and this is the approach used in the scientific papers of Chapters 7 and 9. We refer the reader to Desaulniers et al. (2005) for a detailed description of this method.

We also note that both Appelgren (1969) and Brønmo et al. (2010) have found dynamic column generation more efficient than a priori generation for situations where a priori generation was actually possible. They find the dynamic approach both faster and more flexible as it allows them to deal with larger and more loosely constrained instances than the a priori approach.

In the dynamic column generation approach, the master problem initially only contains a small subset, $\mathcal{R}'^{v}$, of the feasible schedules for each ship $v \in \mathcal{V}$, and new columns that have the potential to improve the current master problem solution are then generated iteratively. The new columns are generated in subproblems, or pricing problems, derived from a Dantzig-Wolfe decomposition, and there are as many independent subproblems as there are ships. Each subproblem corresponds to a specific ship and contains all constraints related to this ship from the original formulation. In each iteration we solve the *restricted master problem* (RMP), which is the linear relaxation of the original master problem though with only its current restricted column set. The dual variables from this solution are then used to guide the generation of promising columns in the subproblems, which are added to the master problem, whereafter a new iteration begins. This process continues until no further promising columns can be generated. At that point, the current solution to the RMP is optimal for the linear relaxation of the original master problem with all feasible schedules in the column set.

Generating the Columns

The purpose of the subproblems is to generate ship specific schedules that can potentially improve the current solution to the RMP. This in turn corresponds to finding feasible schedules with positive reduced costs in the current RMP solution. Since only feasible schedules should be generated, the subproblem must contain all relevant constraints for the corresponding ship, e.g. time windows and capacity constraints.

To illustrate the structure of these subproblems, we let σ_i be the dual variables for constraints (4.2) and (4.3). The variables corresponding to (4.2) are free of sign while the variables corresponding to (4.3) must be nonnegative. Furthermore, we let ω_v be the dual for constraint (4.4) which is also free of sign. Finally, we define $\pi_i = \sigma_i$ for all $i \in \mathcal{N}$ and $\pi_{o(v)} = \omega_v$ corresponding to the origin node. If we restrict our attention to the simple TSRSP formulation presented in Section 2.2, the

corresponding subproblem for ship v is given by:

$$\max \sum_{(i,j)\in\mathcal{A}^v} \left(P_{ij}^v - \pi_i\right)x_{ij}^v \tag{4.7}$$

s.t.

$$(2.4) - (2.9). \tag{4.8}$$

The objective function (4.7) maximises the reduced cost with respect to the current dual variables, and the constraints (2.4)-(2.9) ensure that only feasible schedules are considered. If the optimal schedule has a positive reduced cost, it has the potential to improve the current RMP; hence, it will be represented by a new column in the RMP.

Each subproblem can be modelled as an elementary shortest path problem with time windows (ESPPTW). The ESPPTW has been proven $\mathcal{NP}$-hard in the strong sense by Dror (1994); hence, no pseudo-polynomial algorithms are likely to exist for this problem. When more complex TSRSPs are considered, the subproblems are most often more general elementary shortest path problems with resource constraints (ESPPRC) which are also $\mathcal{NP}$-hard in the strong sense since they generalise the ESPPTW. This is for instance the situation for the multiple cargo case, where, in addition to time, ship capacity is also a constraining resource. The ESPPRC, and thereby also the ESPPTW, entails finding an elementary shortest path between two nodes while satisfying the resource constraints, e.g. time windows and capacity constraints.

In maritime transportation, and in particular within deep sea shipping, voyages travel times are often so long that few, if any, time feasible cycles can be expected in the subproblem networks. Therefore, it is quite common to relax the subproblem to allow non elementary paths, since the regular shortest path problem with resource constraints (SPPRC) can be solved in pseudo polynomial time (see e.g. Desrochers and Soumis (1988); Irnich and Desaulniers (2005)). We use this relaxation in the work presented in Chapters 7 and 9. Note though that if time-feasible cycles do in fact exist, we can generate schedules where the same cargo is transported more than once. In such cases, the solution method must be modified to handle these cycles.

The SPPRC is typically solved by dynamic programming algorithms on the underlying ship specific networks, and we have used this approach in the scientific papers of Chapters 7 and 9. We also see this approach used in e.g. Brønmo et al. (2010) and refer the reader to Desaulniers et al. (1998), Irnich and Desaulniers (2005) and Irnich (2008) for a thorough introduction to the SPPRC, the related dynamic programming algorithms, and several associated concepts.

4.1.4 Full Column Generation Scheme

When a priori column generation has been applied, the number of columns in the master problem is often sufficiently small to allow us to solve the master problem by commercial optimisation software for mixed integer programs and thereby obtain an optimal integral solution. This approach is used in e.g. Fagerholt and Christiansen (2000b) and Fagerholt (2001). When the column count becomes too large for this approach, we must instead solve the linear relaxation of the master problem, and then apply a Branch-and-Bound procedure if the optimal solution if fractional.

When columns have been generated dynamically, once the column generation phase is completed, we have an optimal solution to the linear relaxation of the original master problem in (4.1)-(4.5). If this solution is fractional, the column generation procedure must be embedded in a Branch-and-Bound framework where new columns are generated in each node of the search tree. In this case, the entire procedure is called a Branch-and-Price procedure, since new columns are priced out in each node of the tree.

The strong integer property of the constraint matrix means that the linear relaxation of the master problem will provide very tight upper bounds. When solving the master problem in a Branch-and-Bound framework, we can therefore expect to reach integral solutions after only a few iterations of branching. We verify this property in the scientific paper in Chapter 7, where we incorporate bunker planning in the TSRSP. There, we experience relatively few fractional solutions, and for those fractional solutions we do encounter, the IP gap is very small. Because of these findings, in that paper we do not embed the column generation scheme in a Branch-and-Bound procedure, but simply use commercial optimisation software to solve the integer version

of the final RMP after column generation has been completed. Though this naturally sacrifices optimality, the small IP gaps tells us that this sacrifice is acceptable.

On the other hand, in the scientific paper presented in Chapter 9, where temporal dependencies are added to the master problem, we do embed the column generation scheme in a Branch-and-Bound framework. Due to a relaxation of the temporal dependencies, the upper bound obtained at the root node relates both to the relaxation of binary requirements and to the relaxation of temporal dependencies; hence, the strong integer property of the constraint matrix can no longer guarantee a tight upper bound. Likewise, the added relaxation of the temporal dependencies means that we can no longer count on obtaining a feasible solution after just a few iterations of branching. In fact, we experience extensive branching and must use two different branching schemes to reach feasible solutions: time window branching complemented by constraint branching.

Time window branching was originally presented by Gélinas et al. (1995) as a branching scheme to restore integrality. In Chapter 9, we mainly apply time window branching to deal with relaxed voyage separation requirements and therefore extend the original method to handle these requirements. The basic concept 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 search tree. The key to success with this branching scheme is to select a good time window to split on, and furthermore to select a good split time for this specific time window. This split time must be chosen 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 currently chosen schedule infeasible in each new branch. We note that time window branching is also used in Brønmo et al. (2010).

We illustrate this process in Figure 4.2 for the time window $[a_i, b_i]$ corresponding to a cargo i which is currently transported by two schedules, or twice in a cycle by the same schedule. The start times of these two visits are denoted T_i^1 and T_i^2, respectively. The grey boxes in Figure 4.2 correspond to the so called *feasibility interval* of each of these two visits. These intervals, $[T_i^1, u_1]$ and $[T_i^2, u_2]$, contain all the start times that will allow the corresponding schedule to still remain feasible. This will most often correspond to redistributing waiting time; hence, we have depicted these feasibility intervals as extensions of the current visit times. Since these two intervals are disjoint, we can choose a split time within $(u_1, T_i^2]$, say t_s, and create one branch where the time window for cargo i is $[a_i, t_s - \epsilon]$ and the second visit is infeasible, and one branch where the time window is $[t_s, b_i]$ where the first visit is infeasible. Here, $\epsilon > 0$ is a very small tolerance.

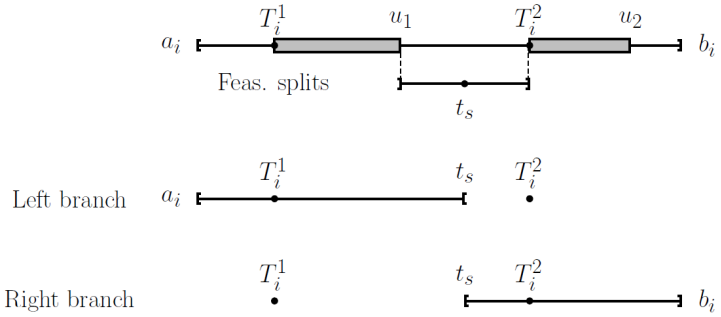


Figure 4.2: Time window branching due to fractionality

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. After branching, all previously generated schedules violating these new time windows, are removed from the master problem in each new branch and the corresponding subproblems are updated to reflect the new time windows.

Constraint branching was introduced by Ryan and Foster (1981), and is an efficient branching scheme developed for solving set partitioning and set packing problems. This branching approach exploits the underlying structure of the constraint matrix by noting that within any optimal fractional solution, there must be at least one pair of constraints that are covered at fractional values. The basic idea is then to enforce that this constraint pair is either covered by different variables,

or covered by the same variable. To give a more formal definition, we denote the two constraints in the fractionally covered constraint pair by c_1 and c_2, and denote the subset of variables that cover both constraints c_1 and c_2 in the constraint matrix by $J(c_1, c_2)$. Reusing our notation from above, we must have

$$0 < \sum_{j \in J(c_1,c_2)} \lambda_j < 1.$$

In the 0-branch, where we enforce the two constraints to be covered by different variables, we then have

$$\sum_{j \in J(c_1,c_2)} \lambda_j = 0.$$

On the other hand, in the 1-branch, where we force the two constraints to be covered by the same variable, we have

$$\sum_{j \in J(c_1,c_2)} \lambda_j = 1.$$

In the 0-branch, all variables in $J(c_1, c_2)$ are removed from the problem, either explicitly or implicitly by simply setting their upper bounds equal to zero. On the other hand, in the 1-branch, all variables covering either constraint c_1 or c_2 but not both, i.e. not in $J(c_1, c_2)$, are removed from the problem.

4.2 Heuristic Solution Approaches

Column generation, as described above, is an exact method. However, as already mentioned, it can easily be modified in a heuristic direction to allow reasonable running times for more difficult problems. This, naturally, leads back to the discussion in Section 3.2 on whether to optimally solve simplified problems or allow the incorporation of additional complexities at the sacrifice of optimality.

In Chapter 7 we integrate bunker planning in the TSRSP, and in this work we follow a heuristic approach by discretising the otherwise continuous bunker purchase variables. It can easily be argued, that at the tactical level, where we are planning months ahead and using simple bunker price forecasts, there is no need to determine the exact amount of bunker to purchase with decimal accuracy. Rather, the main objective is to use the price forecasts to guide the selection of cargoes when we plan routes and schedules, and perhaps to simultaneously negotiate bunker purchases along these routes. The above justifies the discretisation, and we note that this approach of discretising complicating variables is also used in Brønmo et al. (2010), where the continuous cargo quantities are discretised.

The above references are for applications with dynamic column generation, and it is of course also possible to heuristically modify the a priori approach. An example of this can be found in Fagerholt (2001), where the possible times for start of service are discretised, and in Fagerholt and Christiansen (2000a), where the number of a priori generated schedules is reduced by introducing heuristic rules regarding the capacity utilisation of the ships.

In Chapter 8 we explore the tank allocation aspect, and in this work we take an entirely different approach. Whether the overall setup is to use a priori or dynamic column generation or even some sort of local search based method, the generation of new schedules should incorporate the tank allocation aspect to ensure feasibility with respect to stowage. However, at the tactical level, the costs related to tank allocations are insignificant compared to the costs and revenues from carrying cargoes. Therefore, we ignore the cost aspect of the tank allocations and simply focus on feasibility of the allocations. By ignoring the cost aspect, the entire solution method is already of a heuristic nature, though the sacrifice must be considered insignificant. However, we proceed in this direction by splitting the schedule generation phase into two parts: One that ignores the tank allocation aspect and simply generates "standard" schedules, and one that verifies feasibility with respect to stowage for these new schedules. To verify feasibility with respect to stowage, we develop a randomised construction heuristic, modify an earlier optimality-based method, and finally combine these two methods into a hybrid method. This approach can further sacrifice optimality of the entire solution method in two ways:

1. If the hybrid method is unable to find a feasible allocation for a schedule which *does*, however, have a feasible allocation, this schedule is implicitly ignored, even though it could actually be part of the optimal solution.

2. Depending on the procedure for generating columns, splitting the schedule generation into two separate phases can become problematic. Assume, for instance, that columns are generated dynamically using a dynamic programming algorithm where paths are compared iteratively and some are discarded along the way due to domination from other paths. In such a case we can end up with non-dominated paths for which no feasible tank allocation exists, but without having any way of retrieving the otherwise feasible paths that where discarded along the way. This aspect is factored into the solution method in Fagerholt and Christiansen (2000b), where they consider flexible cargo holds. However, this method on the other hand does not incorporate all the operational constraints considered in the work of Chapter 8 such as, for instance, ship stability and regulations for hazard materials.

Chapter 5

Thesis Contribution

The contribution of this thesis is twofold: First, Chapters 2 and 3 contain a thorough introduction to the general area of tramp ship routing and scheduling. They provide the reader with a general understanding of the problem, as well as modeling approaches for it. These chapters also contain review of literature and solution methods within tramp ship routing and scheduling, and, complemented by Chapter 4, provide the reader with a general knowledge on solution approaches for tramp ship routing and scheduling problems. In these chapters we also provide an analysis of both the current status and the future direction of research within the area. Thereby, this part of the thesis can be used to provide researchers, new to the field, with a comprehensive understanding of this research topic.

In Chapter 3 we argue that the current and future direction of research should be within extending the basic TSRSP to incorporate additional complexities. Such extensions will enable more tramp operators to benefit from the solution methods, while simultaneously creating new opportunities for operators already benefitting from existing methods. The second, and main contribution of this thesis, is therefore the exploration of three distinct extensions of the basic TSRSP. The detailed research on these three extensions is provided in three separate scientific papers in Part II of this thesis. We note that one paper considers a novel problem, while the two other papers consider already researched problems, though one paper approaches the known problem from a new perspective. Furthermore, one paper uses an optimality-based solution method, one paper uses both heuristic and optimality-based methods, while one paper explores an exact method. Thereby, these three papers represent a diverse collection of problems and solution methods within tramp ship routing and scheduling. Below we give a short description of the work contained in each of these three papers.

5.1 Incorporating Bunker Planning

Chapter 7, *Tramp Ship Routing and Scheduling With Integrated Bunker Optimization*, contains work on the incorporation of bunker planning into the basic TSRSP. This problem has not previously been considered in the literature.

Most operators manually plan bunkering for their ships. However, considering the significant price variations across bunker ports, and the fact that fuel costs constitute a huge part of daily operating costs, it seems obvious to use optimisation-based tools to plan bunkering. Furthermore, the continuous operation of ships means that they must refuel during their planned schedules; hence, the bunker planning will be affected by the routing and scheduling decisions. Noting that current practice is to separate the two problems by first constructing fleet schedules and then plan bunkering for these fixed schedules, motivates the incorporation of an optimisation-based bunker planning approach into the routing and scheduling phase. The aim of this work is therefore twofold: 1. Provide planners with an optimisation-based planning approach for bunkering, and 2. Incorporate this bunker planning approach into the routing and scheduling phase.

We provide a description as well as a novel mixed integer programming formulation for this new extension of the TSRSP. This model extends standard tramp formulations by incorporating variations in bunker prices, port costs incurred when bunkering, as well as time consumption of

bunkering. For each ship we therefore introduce a number of distinct bunkering options, each corresponding to a specific geographical location with a predefined port cost, bunker price, and time window in which this price is assumed to remain valid. This work has been conducted in collaboration with the Danish shipping company Maersk Tankers A/S and, based on their situation, focuses on full shiploads. This assumption also allows a simple extension of the basic TSRSP to include load dependency on cost, bunker consumption, as well as speed.

We devise a solution method, which simultaneously determines ship routes and schedules, and decides when, where and how much each ship should bunker during its schedule, depending on forward curves for bunker prices. The method relies on column generation with dynamic programming to solve the subproblems. Depending on bunker inventory on arrival at a port, the dynamic programming algorithm dynamically calculates the bunker purchase quantities as partial schedules are extended. The subproblems are solved heuristically by discretising the continuous bunker purchase variables. Results from tuning the devised algorithm indicate that increased refinement of the discretisation resulted in only small increases in the objective function value. Therefore, it seems that solving the continuous version of the problem can at best yield very small improvements in the solutions.

In order to both tune and test the devised algorithm thoroughly, we have also developed instance generators that independently generate cargoes and bunker prices. These instance generators are based on industry data from the collaborating tramp operator and use probability distributions for cargo demand. For our computational study, we generated 150 test instances all with 7 ships and 38 ports. The number of cargoes ranged from 30 to 60 on these instances while the planning horizon ranged from 30 days to 60 days.

Finally, we provide a computational study in which the effect of integrating bunker planning in the routing and scheduling phase is explored through a comparison of the devised solution method with the standard sequential approach, where routes and schedules are planned with no consideration to bunkering. Computational results show that the integrated planning approach can increase profits, and that the decision of which cargoes to carry, and on which ships, is affected by the bunker integration and by changes in the bunker prices.

The main contribution of this work is the exploration of a new version of the tramp ship routing and scheduling problem in which bunker planning is incorporated. This exploration includes a novel mathematical formulation, an efficient solution method, as well as a computational study to evaluate both the performance of the devised method and the effect of incorporating bunker planning. Furthermore, instance generators have been developed for both cargoes and bunker prices, and the generated data as well as the generators can, upon request, be made publicly available.

The work on this project has been presented as follows:

- A paper has been published in EURO Journal on Transportation and Logistics (Vilhelmsen et al., 2013b)
- Presentation at the International Conference on Operations Research (OR2013), Rotterdam, The Netherlands, 2013 (presenter: Charlotte Vilhelmsen)
- A technical report containing the initial work is published at DTU Management Engineering (Vilhelmsen et al., 2013a)

5.2 Determining Tank Allocations

In Chapter 8, *A Heuristic and Hybrid Method for the Tank Allocation Problem in Maritime Bulk Shipping*, we explore the Tank Allocation Problem (TAP) from a tactical perspective. The aim of this work is to develop a new solution method for the TAP, efficient enough to be used as a subproblem solver in an extension of the TSRSP that incorporates the tank allocation aspect.

A major challenge, when operating ships with multiple tanks carrying multiple inhomogeneous products at a time, is how to best allocate cargoes to available tanks. The tank allocations must take into account tank capacity, safety restrictions for onboard cargoes, ship stability and strength, as well as other operational constraints. The allocation problem can be significantly complicated by regulations on hazardous materials, e.g. products in neighboring tanks must be non-reactive

and incompatible products must not succeed each other in a tank unless it is cleaned. Often it is not allowed to move a cargo once it has been allocated and this creates an interdependency between voyage legs. Taking stability, safety restrictions and other operational constraints into consideration it can therefore be extremely difficult, if not impossible, to find a feasible tank allocation for a given set of cargoes. In fact, Hvattum et al. (2009) show that the problem of finding a feasible solution is *NP-Complete*.

The TAP as described above is normally solved at the operational planning level, i.e. *after* fleet schedules have been created. However, separating these two planning problems creates the obvious concern that schedules could potentially be created for which no feasible tank allocation exists. Ideally, the tank allocation aspect should therefore be integrated into the TSRSP for bulk fleets. Since profits from carrying cargoes far outweigh the allocation costs, such as tank cleaning costs, at the tactical planning level we can simplify the tank allocation aspect by ignoring these costs and instead focus on finding a feasible allocation.

The TSRSP is often solved in a way that requires assessment of numerous routes, as for instance in column generation and local search based methods, and for each considered route, the TAP must be solved to assess route feasibility with respect to stowage. Thereby, the TAP must be solved repeatedly; hence, the solution time for the entire procedure will only be acceptable if the TAP can be solved efficiently. This is even more relevant when remembering that, due to uncertainty, routes and schedules are often replanned continuously.

Large and even medium-sized operators can easily have more than 25 ships in their fleets. With multiple cargoes onboard simultaneously, the combinatorial puzzle of assigning cargoes to ships becomes much larger than in the full shipload case and so, 200 routes to evaluate for each ship can easily be a conservative estimate. This yields a total of at least $25 \cdot 200 = 5000$ routes to assess feasibility for. Allowing a run time of up to just 0.25 second means that assessing feasibility with respect to stowage can alone take $0.25 \cdot 5000 = 1250$ seconds, i.e. 21 minutes. Therefore, the requirements for computation are quite strict if we want the method to be applicable to operators of all sizes.

Hvattum et al. (2009) attempt to solve the TAP using constraint programming but find that it fails, primarily due to the stability constraints. They also present a mixed integer formulation for the problem and solve it with a commercial solver. They report running times that are far beyond acceptable in a tactical setting, and they specifically advocate for development of a heuristic method for determining feasibility of the TAP. We have updated and modified their method, which significantly improves their results. Even so, running times are still a bit too long. Neo et al. (2006) present an integer programming model for routing a fleet of multi-compartment tankers with tank allocation incorporated. They solve it using a commercial solver, but even for a single ship with 10 tanks and 10 potential cargoes, running times are above 18,000 seconds. Therefore, in order to facilitate the incorporation of the tank allocation aspect in the TSRSP, the aim of this work has been the development of a heuristic method for efficiently finding feasible cargo allocations for given ship routes.

Our heuristic iteratively allocates the cargoes one by one based on a priority ordering, which is derived from the ratio of the volume of each cargo and the amount of available tank capacity for this cargo. After allocating a cargo, tank availability is updated for all affected cargoes whereby their priority measure is updated and a new iteration can start. We initially only reserve sufficient tank capacity for one cargo at a time, and if we manage to find sufficient capacity for all cargoes, we check if there exists a combination of cargo amounts to tanks that can secure ship stability during the entire route. This is done by solving a simple linear program where each cargo is predefined to be allocated to a given set of tanks.

We allocate each cargo as stable as possible by iteratively choosing tanks that have moment arms in the opposite direction of the ships current stability and strength estimates. Note however, that since cargoes can outbalance each other, we do not *require* each cargo to be allocated in a stable manner on its own but simply *seek* to find a stable allocation for each cargo.

If a cargo in conflict with other cargoes is scattered all over the ship, a lot of neighboring tank capacity becomes unavailable for cargoes in conflict with this cargo. Therefore, we use a crude estimate of tank groupings in order to try to confine conflict cargoes to smaller groups of tanks.

We introduce randomness by allowing the procedure for selecting the next cargo to allocate, and the procedure for selecting tanks for the selected cargo, to sometimes discard the otherwise

deterministically chosen cargo or tank. Each time we allocate a cargo, we iteratively reallocate it until the capacity utilisation is sufficiently high, or until we reach the number of allowed reallocations. In the latter case, we use the best allocation found so far. Within a given time limit, we also allow the heuristic to completely restart each time it fails to find a feasible allocation.

Our computational study is based on 486 instances from an instance generator developed in Hvattum et al. (2009). These instances are based on two real tank ships with, respectively, 24 tanks and 38 tanks. The generated instances have between 20 and 40 cargoes from three different cargo types. Computational results show that our heuristic can solve 99% of the considered instances within 0.4 seconds and all of them if allowed more time. The heuristic does however struggle on two instances causing an overall longer average running time than found with the updated optimality-based method from Hvattum et al. (2009). However, on the remaining 484 instances our heuristic is much faster than both the updated and modified version of the optimality-based method from Hvattum et al. (2009).

The two instances that cause the heuristic trouble are actually solved relatively quickly by the modified version of the optimality-based method while this method instead struggles with other instances. Therefore, we have combined the two methods to obtain an even faster hybrid method that first runs the heuristic for 0.2 seconds and if no feasible solution is found, then runs the modified optimality-based method on the parts of the instance that the heuristic has not solved. The results from running this hybrid method shows that it cuts between 90% and 94% of average running times compared to the three other presented algorithms. In fact, no matter the allowed time limit, no other method solves more instances than the hybrid method, and the average running time for the hybrid method is just 0.027 second. Thereby, we have developed a method that is efficient enough to facilitate the incorporation of the tank allocation aspect in the TSRSP.

The main contribution of this work is a new and heuristic solution method for the TAP. This heuristic creates robust solutions and is flexible enough to incorporate operational considerations such as ballast tanks and moving cargoes between tanks after allocation. It is very fast and clearly outperforms previous methods on 484 of the 486 considered instances. A further contribution of this work is the construction of a hybrid method that combines our heuristic with an earlier optimality-based method, which we have modified to improve efficiency. This hybrid method clearly outperforms all other methods proposed, and solves all problem instances with an average running time of just 0.027 second. Thereby, we have a solution method efficient enough to allow the incorporation of tank allocations into the basic TSRSP.

The work on this project has been presented as follows:

- A paper has been submitted for publication in a relevant international journal (Vilhelmsen et al., 2014a)
- Presentation at the 4th International Conference on Computational Logistics (ICCL'13), Copenhagen, Denmark, 2013 (presenter: Charlotte Vilhelmsen)
- Presentation at the 3rd International Symposium on Combinatorial Optimization (ISCO2014), Lisbon, Portugal, 2014 (presenter: Charlotte Vilhelmsen)
- A technical report containing this work is published at DTU Management Engineering (Vilhelmsen et al., 2014b)

5.3 Incorporating Voyage Separation Requirements

In Chapter 9, *Tramp Ship Routing and Scheduling with Voyage Separation Requirements*, we consider the TSRSP with voyage separation requirements (VSRs), and devise a new, exact solution method for this problem.

In this work, the basic TSRSP is extended to incorporate VSRs that enforce minimum time spreads between specific voyages. This is one way of balancing the conflicting objectives of maximising profits for the tramp operator and minimising inventory costs for the charterer, since these costs increase if consecutive voyages are not performed with some separation in time. In this respect, the incorporation of VSRs correspond to a crude way of viewing the TSRSP in the broader context of the supply chain. Norstad et al. (2013) show that the incorporation of VSRs

can significantly improve the spread of the voyages at only marginal profit reductions. Thereby, the tramp operator can at very little cost improve customer service for the charterer, and in a market where competition is tough and freight rates are already low, this customer satisfaction aspect is interesting.

Computational results from Norstad et al. (2013) show that their two methods are not applicable for larger and more complex problem instances. The aim here is, therefore, to develop a more efficient solution method.

We present a new mixed integer programming formulation for this problem and develop a new, exact solution method for it. This method is a Branch-and-Price procedure based on a Dantzig-Wolfe decomposition of the original formulation. In the master problem, the VSRs are relaxed along with the binary variable restrictions. We present a new time window branching scheme that can restore feasibility with respect to the VSRs, and to some extent also restore integrality. Since this branching scheme is not complete with respect to integrality, we complement it by constraint branching, which utilises the strong integer properties of the master problem constraint matrix, to efficiently eliminate fractionality. We use a dynamic programming algorithm to solve the subproblems and thereby generate the master problem columns.

We provide a computational study based on 16 data instances of varying complexity and size. These data instances have been generated by the test instance generator described in Norstad et al. (2013), which is based on industry data. On these instances, the fleet size ranges from 10 to 32 ships, the number of voyages ranges from 10 to 64, while the planning horizon is between 90 and 150 days. Computational results show that our algorithm finds very good if not optimal solutions extremely fast, although one instance requires longer time. To properly evaluate the performance of our algorithm, we compare it to an earlier a priori path generation (APPG) method, which is the most efficient method from Norstad et al. (2013). This method first a priori generates feasible paths and then uses a commercial solver to solve the path flow formulation containing these generated paths. We show that, due to a domination test in the path generation phase, this APPG method is not exact. Computational results confirm this, since our algorithm consistently finds solutions that are equally good or better than those from the APPG method. In fact, the profit increase from using our algorithm compared to the APPG method is as high as 6% for one instance. Furthermore, for all but one instance, our solutions are obtained in the same or shorter time than what the APPG method uses.

The main contribution from this work is the development of a new, exact method for the TSRSP with voyage separation requirements, which relies on a tailor made time window branching scheme. This method is able to find very good if not optimal solutions extremely fast. A further contribution of this work is the comparison with the APPG method, which we show is not an exact method. Our computational study confirm this as our method consistently finds equal or better solutions than this other method. Furthermore, on all but one instance, our solution is found in equal or shorter time than that of the APPG method.

The work on this project has been presented as follows:

- A technical report containing this work is published at DTU Management Engineering (Vilhelmsen and Lusby, 2014)
- A paper will be submitted for publication in a relevant international journal after approval from collaborating partners (Vilhelmsen and Lusby, 2015)

Chapter 6

Conclusion

Aside from acquiring and promoting general knowledge of tramp ship routing and scheduling, the main aim of this PhD project was to develop new mathematical models and solution methods for problems within this area. We have done so through three distinct research projects each described in details in Chapters 7-9.

In Chapter 7 we consider a problem new to the research field and propose a novel mixed integer programming formulation as well as an optimality-based solution method based on column generation. We show that the incorporation of bunker planning into the routing and scheduling phase can increase profits for tramp operators. For the considered problem instances, the profit increase is moderate. However, we suspect that this is due to the geographical setup of these instances as well as the relatively low magnitude of fluctuations in bunker prices on these instances. It could therefore be very interesting to extend this project by exploring other data instances. Furthermore, it could be interesting to extend the model to allow price uncertainty and apply stochastic programming to solve it.

In Chapter 8 we explore an operational planning problem already known from the literature, though on a tactical level. We develop a new heuristic method for the Tank Allocation Problem and also create a hybrid method from the combination of this heuristic and a previously presented optimality-based method, which we have modified to improve efficiency. This hybrid method outperforms all previous methods and is efficient enough to facilitate the incorporation of tank allocations into the routing and scheduling phase. It would, however, be very interesting to improve this method to allow flexible cargo sizes, since most operators involved in liquid bulk shipping face this situation.

In Chapter 9 we consider a problem previously presented in literature and present a new, exact method for it. This problem incorporates voyage separation requirements in order to balance the conflicting objectives of profit maximisation for the tramp operator and cost minimisation for the charterer, whose inventory costs increase if consecutive voyages are not performed with some separation in time. This problem therefore acts as an initial attempt of viewing the TSRSP in the broader context of a supply chain. Our solution method is a Branch-and-Price procedure, which is able to find very good if not optimal solutions extremely fast. A comparison with an earlier method from the literature, which we show is not an exact method, shows that our algorithm consistently finds equal or better solutions than this earlier method. Furthermore, on all but one instance, our solution is found in equal or shorter time than that of the earlier method. On four instances, our algorithm is unable to prove optimality within the time limit. Although the integrality gap is small for all four instances, it would still be interesting to explore different modifications to the branching procedure.

6.1 Main Contributions

Overall, we have explored three distinct extensions of the basic TSRSP, and through this work developed both new models and methods for tramp ship routing and scheduling. This thesis makes the following contributions to tramp ship routing and scheduling research:

- A description and mathematical formulation for the TSRSP with integrated bunker optimisation, a problem not previously considered in tramp shipping literature.
- An optimality-based solution method for the above problem.
- Test instance generators for both cargoes and bunker prices, which can be made publicly available upon request.
- A heuristic method for finding feasible tank allocations.
- A hybrid method that combines the above heuristic with an earlier optimality-based method, which we have modified to improve efficiency. This hybrid is very fast and facilitates the incorporation of tank allocations into the basic TSRSP.
- An analysis of an earlier method for the TSRSP with voyage separation requirements. This analysis shows that this earlier method is not an exact method as otherwise stated in the literature.
- A new, exact, and more efficient method for the TSRSP with voyage separation requirements.

6.2 Future research directions

Through this PhD study we have gained a general knowledge of existing research within tramp ship routing and scheduling. From this knowledge it is obvious that one of the main directions of future research lies in the development of models and methods for extensions of the basic TSRSP rather than in the development of new methods for the basic problem. Although some extensions, such as variable speed, are quite generic, a lot of extensions will be operator specific. Therefore, it is hard to point in any one direction for future research.

With this being said, we do however want to mention three interesting research topics that, in our opinion, deserve great attention in future research within tramp shipping:

- We find that the static and deterministic approach in most work on tramp shipping constitutes a huge simplification for most operators. Naturally, one must walk before one can run, but now that the basic TSRSP can be solved efficiently, it seems obvious to expand the research to include both dynamic and stochastic models. In fact, reality for most operators is that new spot cargoes are continuously revealed, and in many cases the exact information on these cargoes is not known in advance. E.g. the destination port can be unknown. The ongoing optimisation means that the solution methods should either be embedded in a rolling horizon approach or incorporate a value of time to account for schedules that occupy ships well into the next planning period. Both these approaches should then utilise information on future cargoes in a stochastic manner.
- As already mentioned, the work in Chapter 9 in some sense relates the TSRSP to the broader context of a supply chain. In industrial shipping, where the ship operator is also the cargo owner, the supply chain aspect seems obvious and much literature in this area also includes this aspect, see (Christiansen et al., 2013). Even though this aspect seems less obvious in a tramp shipping setting, it is still a very interesting topic for further research, and we see two main reasons for this. First, in a market with tough competition and freight rates already critically low, it seems obvious for tramp operators to instead improve their customer service to attract more business. Second, optimising over the entire supply chain, or at least parts of it, rather than just the isolated TSRSP, will no doubt enable overall cost reductions. With the right collaboration setup, some part of these cost reductions will reflect back on the tramp operator and allow an increase in profit. Furthermore, we note that including information from the supply chain will also give a better understanding of future demand and thereby assist in resolving the issue from above regarding dynamic and stochastic modelling approaches.

- Finally, during this PhD study, a considerable amount of time has been devoted to both gathering and generating data that accurately reflects the situation in industry. This time could obviously have been devoted to other research topics if data had been readily available. This strongly motivates the development of benchmark data for the tramp shipping community just as we see it for vehicle routing and as has recently been developed for the liner shipping community (Brouer et al., 2014). Aside from enabling more research within the area, such benchmark instances would also facilitate easy comparison of solution methods developed for similar problems. We note that, upon request, the instance generators developed for the bunker project and presented in details in Chapter 7 can be made publicly available.

Bibliography

H. Andersson, M. Christiansen, and K. Fagerholt. The maritime pickup and delivery problem with time windows and split loads. *INFOR*, 49(2):79–91, 2011a.

H. Andersson, J.M. Duesund, and K. Fagerholt. Ship routing and scheduling with cargo coupling and synchronization constraints. *Computers and Industrial Engineering*, 61(4):1107–1116, 2011b.

L.H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3:53–68, 1969.

L.H. Appelgren. Integer programming methods for a vessel scheduling problem. *Transportation Science*, 5:64–78, 1971.

G. Brønmo, M. Christiansen, K. Fagerholt, and B. Nygreen. A multi-start local search heuristic for ship scheduling - a computational study. *Computers & Operations Research*, 34:900–917, 2007a.

G. Brønmo, M. Christiansen, and B. Nygreen. Ship routing and scheduling with flexible cargo sizes. *Journal of the Operational Research Society*, 58(9):1167–1177, 2007b.

G. Brønmo, B. Nygreen, and J. Lysgaard. Column generation approaches to ship scheduling with flexible cargo sizes. *European Journal of Operational Research*, 200(1):139–150, 2010.

B.D. Brouer, J.F. Alvarez, C.E.M. Plum, D. Pisinger, and M.M. Sigurd. A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48 (2):281–312, 2014.

E.R. Butchers, P.R. Day, A.P Goldie, S. Miller, J.A. Meyer, D.M. Ryan, A.C. Scott, and C.A. Wallace. Optimized crew scheduling at air new zealand. *Interfaces*, 31(1):30–56, 2001.

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

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

M. Christiansen and B. Nygreen. A method for solving ship routing problems with inventory constraints. *Annals of Operations Research*, 81:357–378, 1998.

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

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

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

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

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

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

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

J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Routing. Handbooks in Operations Research and Management Science, vol. 8*, chapter 2, pages 35–139. North-Holland, Amsterdam, 1995.

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

K. Fagerholt. Ship scheduling with soft time windows: An optimisation based approach. *European Journal of Operational Research*, 131(3):559–571, 2001.

K. Fagerholt. A computer-based decision support system for vessel fleet scheduling - experience and future research. *Decision Support Systems*, 37(1):35–47, 2004.

K. Fagerholt and M. Christiansen. A combined ship scheduling and allocation problem. *Journal of the Operational Research Society*, 51:834 – 842, 2000a.

K. Fagerholt and M. Christiansen. A travelling salesman problem with allocation, time window and precedence constraints - an application to ship scheduling. *International Transactions in Operational Research*, 7(3):231 – 244, 2000b.

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

K. Fagerholt, L.M. Hvattum, T.A.V. Johnsen, and J.E. Korsvik. Routing and scheduling in project shipping. *Annals of Operations Research*, pages 1–15, 2011.

R.A. Gatica and P.A. Miranda. Special issue on latin-american research: A time based discretization approach for ship routing and scheduling with variable speed. *Networks and Spatial Economics*, 11(3):465–485, 2011.

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

L.M. Hvattum, K. Fagerholt, and V.A. Armentano. Tank allocation problems in maritime bulk shipping. *Computers & Operations Research*, 36(11):3051–3060, 2009.

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

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

J.N. Jung, M.N. Kang, H.R. Choi, H.S. Kim, B.J. Park, and C.H. Park. Development of a genetic algorithm for the maritime transportation planning of car carriers. *Dynamics in logistics*, 6: 481–488, 2011.

K. Kang, W.-C. Zhang, L.-Y. Guo, and T. Ma. Research on ship routing and deployment mode for a bulk. *International Conference on Management Science and Engineering - Annual Conference Proceedings*, pages 1832–1837, 2012a.

M.H. Kang, H.R. Choi, H.S. Kim, and B.J. Park. Development of a maritime transportation planning support system for car carriers based on genetic algorithm. *Applied Intelligence*, 36(3): 585–604, 2012b.

S.-H. Kim and K.-K. Lee. An optimization-based decision support system for ship scheduling. *Computers and Industrial Engineering*, 33(3-4):689–692, 1997.

K. Kobayashi and M. Kubo. Optimization of oil tanker schedules by decomposition, column generation, and time-space network techniques. *Japan Journal of Industrial and Applied Mathematics*, 27(1):161–173, 2010.

J.E. Korsvik and K. Fagerholt. A tabu search heuristic for ship routing and scheduling with flexible cargo quantities. *Journal of Heuristics*, 16(2):117–137, 2010.

J.E. Korsvik, K. Fagerholt, and G. Laporte. A tabu search heuristic for ship routing and scheduling. *Journal of the Operational Research Society*, 61(4):594–603, 2010.

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

S.A. Lawrence. *International Sea Transport: The Years Ahead*, pages 1–316. Lexington Books, Lexington, MA., 1972.

D.-Y. Lin and H.-Y. Liu. Combined ship allocation, routing and freight assignment in tramp shipping. *Transportation Research Part E: Logistics and Transportation Review*, 47(4):414–431, 2011.

F. Malliappi, J. A. Bennell, and C. N. Potts. A variable neighborhood search heuristic for tramp ship scheduling. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6971 LNCS:273–285, 2011.

C. Michelsen. Emissions trading scheme for the shipping industry? `http://s.tt/15q5B`, January 2012.

I.K. Moon, Z.B. Qiu, and J.H. Wang. A combined tramp ship routing, fleet deployment, and network design problem. *Maritime Policy and Management*, 2014. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84891807001&partnerID=40&md5=46e36df2bcec237075f164ea3de0b21b`.

K.-H. Neo, H.-C. Oh, and L.A. Karimi. Routing and cargo allocation planning of a parcel tanker. In *16th European Symposium on Computer Aided Process Engineering and 9th International Symposium on Process Systems Engineering*, pages 1985 – 1990, 2006.

I. Norstad, K. Fagerholt, and G. Laporte. Tramp ship routing and scheduling with speed optimization. *Transportation Research Part C: Emerging Technologies*, 19(5):853–865, 2011.

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

H.-C. Oh and I.A. Karimi. Routing and scheduling of parcel tankers: a novel solution approach. In A. Bruzzone, F. Longo, Y. Merkuriev, G. Mirabello, and M.A. Piera, editors, *The 11th International Workshop on Harbor Maritime Multimodal Logistics Modeling and Simulation*, pages 98 – 103, September 2008.

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

C.E.M. Plum, D. Pisinger, and P.N. Jensen. Bunker purchasing in liner shipping. 2014, forthcoming.

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

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

D. Ronen. Marine inventory routing: Shipments planning. *Journal of the Operational Research Society*, 53(1):108–114, 2002.

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

J. Schönberger, H. Kopfer, and D.C. Mattfeld. A combined approach to solve the pickup and delivery selection problem. In U. Leopold-Wildburger, F. Rendl, and G. Wäscher, editors, *Operations Research Proceedings 2002*, pages 150–155. Springer, 2003. Selected Papers of the International Conference on Operations Research (SOR 2002), Klagenfurt, September 2-5, 2002.

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

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

P. Toth and D. Vigo. *The vehicle routing problem.* SIAM, 2002.

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

C. Vilhelmsen and R.M. Lusby. Tramp ship routing and scheduling with voyage separation requirements. *Technical Report, Department of Management Engineering, Technical University of Denmark*, 2014.

C. Vilhelmsen and R.M. Lusby. Tramp ship routing and scheduling with voyage separation requirements. *To be submitted to relevant international journal*, 2015.

C. Vilhelmsen, R.M. Lusby, and J. Larsen. Routing and scheduling in tramp shipping - integrating bunker optimization. *Technical Report, Department of Management Engineering, Technical University of Denmark*, 2013a.

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

C. Vilhelmsen, J. Larsen, and R.M. Lusby. A heuristic and hybrid method for the tank allocation problem in maritime bulk shipping. *Submitted to relevant international journal*, 2014a.

C. Vilhelmsen, J. Larsen, and R.M. Lusby. A heuristic and hybrid method for the tank allocation problem in maritime bulk shipping. *Technical Report, Department of Management Engineering, Technical University of Denmark*, 2014b.

Part II

Scientific Papers

Chapter 7

Tramp Ship Routing and Scheduling with Integrated Bunker Optimization

Charlotte Vilhelmsen Richard M. Lusby Jesper Larsen

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

Abstract A tramp ship operator often has contracted cargoes that must be carried and seeks to maximize profit by carrying optional cargoes. Hence, tramp ships operate much like taxies following available cargo and not according to fixed route networks and itineraries as liner ships. Marine fuel is referred to as *bunker* and bunker costs constitute a significant part of daily operating costs. There can be great variations in bunker prices across ports so it is important to carefully plan bunkering for each ship. As ships operate 24 hours a day, they must refuel during operations. Therefore, route and schedule decisions affect the options for bunkering. Current practice is, however, to separate the two planning problems by first constructing fleet schedules and then plan bunkering for these fixed schedules. In this paper we explore the effects of integrating bunker planning in the routing and scheduling phase for a tramp operator sailing full shiploads, i.e. carrying at most one cargo onboard each ship at a time. We present a mixed integer programming formulation for the integrated problem of optimally routing, scheduling and bunkering a tramp fleet carrying full shiploads. Aside from bunker integration, this model also extends standard formulations by using load dependent costs, speed and bunker consumption. We devise a solution method based on column generation with a dynamic programming algorithm to generate columns. The method is heuristic mainly due to discretization of the continuous bunker purchase variables. We show that the integrated planning approach can increase profits and that the decision of which cargoes to carry and on which ships is affected by the bunker integration and by changes in bunker prices.

7.1 Introduction

It is estimated that over 80% of world trade is carried by the international shipping industry (UNCTAD, 2011) and world trade therefore depends on the industry's efficiency and competitive freight rates. Hence, research to increase efficiency within maritime transportation is important, and, taking the mere size of this huge industry into consideration, even small improvements can have great impact.

An important part of utilizing the existing fleet efficiently is routing and scheduling the ships, i.e. assigning cargoes to ships while simultaneously finding the sequence and timing of port calls for all ships. Many ship operators use experienced planners to manually route and schedule the

fleet. However, increased competition and recent trends of mergers among, and pooling of, shipping companies have increased the pressure as well as the difficulty of devising efficient schedules manually due to the increased fleet sizes (Christiansen et al., 2004). Therefore, there is a need for an automated approach to the planning that can both aid the construction of efficient schedules and enable fast changes to existing schedules in case of new opportunities or changed demand.

In this paper we focus on tramp shipping where ships operate much like taxies following the available cargoes and not according to a fixed route network and itinerary as in liner shipping. Routing and scheduling within tramp shipping is therefore a more dynamic and ongoing process compared to that of liner shipping. A tramp operator often has some contracted cargoes that must be carried and seeks to maximize profit by carrying optional cargoes found in the spot market.

Tramp ship routing and scheduling is very closely related to the well researched Vehicle Routing Problem (VRP) and its variants but there are, however, important differences that facilitate the development of industry specific methods. To mention a few, we note that optional cargoes are not considered in the standard VRP and that ships pay port fees and operate around the clock.

Marine fuel is also referred to as *bunker fuel* or simply *bunker* while refueling is called *bunkering*. Fuel costs constitute a significant part of daily operating costs and since bunker prices can vary significantly across ports, it is important to carefully plan the bunkering of each ship. The recent increase in oil prices adds further motivation for operators to plan bunkering optimally, yet many still use manual planning. Ships operate 24 hours a day so they must refuel during operations. Hence, route and schedule decisions will affect the options for bunkering. Consequently, it seems natural to integrate bunker planning in the routing and scheduling phase and consider the combined routing, scheduling and bunkering problem. Current practice is, however, to separate the two problems by first constructing fleet schedules and then plan bunkering for these fixed schedules.

In this paper we explore the effects of integrating bunker planning in the routing and scheduling phase for a tramp operator sailing full shiploads, i.e. carrying at most one cargo onboard each ship at a time. We present a mixed integer programming formulation for the integrated problem of optimally routing, scheduling and bunkering a tramp fleet carrying full shiploads. This model extends standard tramp formulations by accounting for bunkering time, variations in bunker prices and bunker ports costs and further by using load dependent costs as well as speed and bunker consumption. We devise a heuristic solution method that can simultaneously select which optional cargoes to carry, how cargoes should be allocated to ships, determine ship routes and schedules, and decide when, where and how much each ship should bunker during its schedule depending on forward curves for bunker prices. The method relies on column generation with a dynamic programming algorithm to generate columns. Computational results show that this integrated planning approach can increase profits, and that the decision of which cargoes to carry and on which ships is affected by the bunker integration and by changes in the bunker prices.

The remainder of the paper is organized as follows. In Section 7.2 relevant literature is presented. Section 7.3 provides a problem description as well as a mathematical model for the problem, while the devised solution method is described in Section 7.4. Section 7.5 describes some instance generators that we have developed to acquire necessary data on cargoes and bunker prices. In Section 7.6 we tune the devised algorithm and in Section 7.7 we explore the effects of integrating bunker planning in the routing and scheduling phase through a comparison of the integrated approach and the sequential approach. We also investigate the method's sensitivity to bunker prices. Finally, concluding remarks and suggestions for future work are discussed in Section 7.8.

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 review papers, Ronen (1983), Ronen (1993), Christiansen et al. (2004) and Christiansen et al. (2013).

Recent work on tramp ship routing and scheduling include Kang et al. (2012) who consider the interaction between ship routing and scheduling and ship deployment, and Stålhane et al. (2012) who investigate split loads. Fagerholt and Ronen (2013) present and consolidate results for three practical extensions within bulk shipping: (1) flexible cargo quantities, (2) split cargoes, and (3)

sailing speed optimization, while Tang et al. (2013) consider speed optimization.

The tramp ship routing and scheduling problem is closely related to vehicle routing problems. Most similar to our problem is the vehicle routing problem with pickup and deliveries and time windows (VRPPDTW) for which we refer the reader to Desaulniers et al. (2002). There are, however, important differences between the maritime version of the problem and the land based one, creating the need for tailor made models and solution methods for each industry. Ronen (1983), Ronen (2002) and Christiansen et al. (2004) elaborate on these differences but to mention a few, we note that ships pay port fees and operate continuously. Hence, ships have different starting positions and starting times, as some ships can be occupied with prior tasks when planning begins. Even in multi-depot versions of the VRPPDTW vehicles must return to their home depot whereas ships do not have to return to their starting point. Finally, the distinction between contract cargoes and optional cargoes leads to a priority on cargoes not used in VRPPDTW where all customers must be serviced at minimum cost. In contrast, the tramp objective is to maximize profit as in the less known Pickup and Deliver Selection Problem, see Schönberger et al. (2003).

Column generation has received much attention and achieved great success for vehicle routing but is not as frequently used within maritime transportation. This is partly because the large number of constraints reduce the solution space to such an extent that - combined with the major uncertainty within maritime transportation - feasible schedules only consist of a few voyages. Hence, all feasible combinations can be enumerated and, so, it is often sufficient to apply a priori column generation. However, within tramp ship scheduling both Appelgren (1969) and Brønmo et al. (2010) have found dynamic column generation more efficient than a priori generation. Brønmo et al. (2010) report that in their experiments, the dynamic approach is both faster and enables them to deal with larger or more loosely constrained instances than a priori generation. In line with that, recent years have shown an increase in maritime papers that explore dynamic column generation, see e.g. Kobayashi and Kubo (2010), Hennig et al. (2012) and Stålhane et al. (2012). Furthermore, Desaulniers et al. (2005) devoted two whole chapters to column generation in maritime problems.

Within vehicle routing research on refueling policies can be found in Lin et al. (2007) and Lin (2008). Software products for refueling, called *fuel optimizers*, have also been developed for the trucking industry. Suzuki (2008) gives a description of such systems and a literature review while Suzuki and Dai (2012) discuss solution methods. These systems use the latest price data to calculate which truck stops to use and how much to purchase at each stop to minimize refueling costs. The above work on refueling policies are for single vehicles traveling on fixed routes. Hence, there is no integration of refueling with routing and scheduling. Also, since customers have already been assigned to vehicles, the interdependency of vehicle routes is ignored and the problem is decomposed into independent one-vehicle problems while we must consider the entire fleet.

Within air transportation work on refueling policies can be found in Darnell and Loflin (1977), Stroup and Wollmer (1992), Abdelghany et al. (2005) and Zouein et al. (2002). However, they also consider refueling policies for fixed routes and do not allow aircraft to divert from their routes for refueling. In fact, since routes are fixed, the refueling policy problem relates more to liner shipping where the combinatorial aspect, i.e. the route selection, from tramp shipping is not present.

Within liner shipping Yao et al. (2012) explore refueling policies where sailing speed is a decision variable. As they consider a liner service, they too assume a fixed route and do not allow diversions from routes to refuel. Similarly, Besbes and Savin (2009) also explore refueling policies for liner ships for fixed routes. In contrast, Notteboom and Vernimmen (2009) consider the impact of increasing bunker prices on the actual design of liner services. For a single liner vessel on a fixed route Kim et al. (2012) seek to minimize the bunker related costs together with the cost of the ship's time and environmental costs. They determine the optimal ship speed, bunkering ports, and amounts of bunker fuel. Finally, Wang et al. (2013) present a review on solution methods for bunker consumption optimization problems within liner shipping and propose several new ones.

Within tramp shipping, as far as we know, only two papers address optimal refueling. Oh and Karimi (2010) consider a multi parcel tanker and propose a mixed integer programming model to optimize bunkering. They use speed as a decision variable and take uncertainty of fuel prices into account but they too assume a fixed route. In contrast, Besbes and Savin (2009) simultaneously optimize routes and bunker plans. However, their approach is much more strategic than ours. They formulate the problem as a stochastic dynamic program. They only consider one ship and do not consider actual cargoes. Instead they assume a stochastic revenue process that leads them to

explore optimal network *cycles*. In fact, they view the problem more as an inventory management problem with no end, and, consequently, seek to maximize *long term average* profit. They characterize the optimal refueling policy when prices are constant over time and do not differ across ports and when prices are constant over time but differ across ports. However, they do not consider the case where prices vary over time and at the same differ across ports as we do.

7.3 Problem Description

In this section we give a problem description starting with the pure tramp ship routing and scheduling problem. We then move on to include bunkering and present a mathematical model for the Tramp Ship Routing And Scheduling Problem with Bunker Optimization (TSRSPBO).

7.3.1 The Pure Tramp Ship Routing and Scheduling Problem

A tramp operator has long term contracts that obligates him to carry some cargoes and can choose to carry additional cargoes, so called *spot cargoes*, if fleet capacity allows it and it is profitable. The objective is to create a profit maximizing set of fleet schedules, one for each ship, where a schedule is a sequence and timing of port calls representing cargo loading and discharging. The optimal solution therefore combines interdependent decisions on which optional cargoes to carry, the assignment of cargoes to ships and the optimal sequence and timing of port calls for each ship.

A cargo is mainly characterized by the quantity to be transported, the revenue obtained from transporting it and the pickup and discharge port. There is also a service time for loading and discharging and a time window giving the earliest and latest start for loading. In some cases there is also a time window for discharge. A tramp fleet is usually heterogeneous, comprised of ships of different sizes, load capacities, bunker consumptions, speeds, and other characteristics. Ships can be occupied with prior tasks when planning starts so each ship is further characterized by the time it is available for service and the location it is at when it becomes available. The characteristics of a ship determine which cargoes, ports and canals it is compatible with.

As we consider a fixed fleet, we disregard fixed setup costs and focus on variable operating costs. The main sailing cost is fuel cost and this is different for each ship and load dependent. In traditional tramp ship routing and scheduling models, sailing cost, and in turn bunker consumption, is assumed independent of the load of the ship. We, however, do not make this assumption. When loading and discharging, ship dependent port costs are incurred, and ships also consume bunker although much less than at sea. Other operator specific costs can also be relevant.

The research presented here has been conducted in collaboration with the Danish shipping company Maersk Tankers A/S involved in, among other things, transportation of refined oil products worldwide. Based on their case we focus on full shiploads, i.e. each ship carries at most one cargo at a time. When considering full shiploads and not including bunkering, pickup and delivery of a cargo must be performed directly after each other and, hence, the two tasks can be aggregated. This yields a simple model and we refer the reader to Christiansen et al. (2007) for a mathematical arc flow formulation for the pure tramp ship routing and scheduling problem with full shiploads.

7.3.2 Incorporating Bunker

The model presented in Christiansen et al. (2007) for the pure routing and scheduling problem, just as most other tramp ship routing and scheduling models, assumes fixed sailing costs with no consideration to the great variations in bunker prices, the port costs incurred when bunkering, or the time consumption of bunkering. In our work, we integrate considerations for bunker price variations, bunker port costs as well as the time aspect of bunkering and extend the pure routing and scheduling formulation to include variables for bunker purchases, new constraints to incorporate these variables and, finally, an extended objective function that reflects this new way of calculating bunker costs, and in turn sailing costs. We also incorporate load dependent bunker consumption. Bunker consumption also depends on the ship's speed. However, in reality speed is not necessarily fixed but instead allowed to vary with the load of the ship. E.g. if a ship sails at 'full speed', the actual speed depends on the ship's load. Likewise, a speed setting often used is 'ECO speed', i.e. most economical speed, and this speed also depends on the ship's load. We assume each ship sails

at 'ECO speed' and so, the actual speed and bunker consumption depend on the load of the ship. Likewise, we allow costs to be load dependent which is often the case for port and canal costs.

Bunkering can take place at a bunker port where ships enter port just to refuel or at a pickup or discharge port since almost all ports involved in shipping also sell bunker. Bunkering at a pickup or discharge port has the obvious benefit of avoiding detours just for bunker, only incurring port costs once and even saving time if concurrent bunkering is allowed. However, price variations between ports can easily be large enough to compensate for the extra cost of a detour, the extra port costs, and also the extra time consumption. In fact, a few ports seem to dominate bunker sales because of their strategic location along major trade routes thereby limiting the detours necessary for ships to refuel there. Two examples of such ports are Malta and Singapore (Oh and Karimi, 2010).

A bunker option is mainly characterized by its geographical location, port costs, bunker price and time window in which this price is assumed to remain valid. Several bunker options can represent the same physical bunker port but at different times and, hence, with different prices. Time dependent port costs or port opening hours can also cause a separation of one bunker port into many bunker options with different prices and associated time windows. Due to high volatility in bunker prices these time windows are bound to be narrow and without loss of generality we assume they are so narrow that no ship will use the same bunker option twice. If time windows are not narrow enough for such an assumption, each time window can simply be split into several smaller time windows each with an associated bunker option until the assumption is valid.

Data from the collaborating tramp operator on distances, port costs, fleet bunker consumption and fleet bunker tank capacity shows that it is very reasonable to assume that each ship makes at most one bunker stop in between cargo stops. This assumption is also used by Oh and Karimi (2010) and helps reduce the problem size. However, our solution method also works without this assumption although longer running times must be expected due to the increase in problem size.

Ships operate 24 hours a day so there is no natural end to the optimization problem. Hence, the condition of the fleet at the end of the planning period affects optimization in the next period. With bunker included in the process, the initial bunker inventory for each ship is therefore an important part of fleet data. Likewise, any remaining bunker onboard ships at the end of their schedules must be considered a valuable resource for the next planning period independent of future demand. To account for this resource, we put a premium on any quantity above the initial bunker level for each ship and call this quantity *bonus bunker*. Similarly, ships that end their schedule with less bunker than the initial level must pay for using this resource. As we discuss later, it is in fact vital for our solution method to account for this bonus bunker. However, the actual value of it is difficult to price. Using the price of the last visited bunker port is not possible for ships that due to high initial bunker inventory or idleness did not refuel during their schedule. It also leads to arbitrage since visiting an expensive bunker port last to purchase a small amount will drive up the resell price even though the bonus bunker might have been bought at a cheap bunker port. Likewise, there is no incentive to fill up if a ship passes a cheap bunker port last. Allowing bonus bunker to be resold at an average price of the region that the ship ends its schedule in motivates repositioning ships to regions with high bunker prices with no consideration for future cargo demand. Therefore, we calculate premiums for bonus bunker at an average price of all bunker options with time windows containing the end of the planning horizon, i.e. a geographically independent forecast of the average bunker price at the end of the planning horizon. Ship data will now also include a minimum and maximum bunker level corresponding to a required safety level and tank capacity, respectively.

Concurrent bunkering and loading/discharging can easily be added to the model. If i and j correspond to the same physical port which allows concurrent bunkering and i correspond to a bunker option while j correspond to a cargo node, then T^v_{ij} and T^v_{ji} will not contain time for bunkering (assuming bunkering time is less than loading/discharging time). We have chosen not to include it in our analysis and further assume that all bunker options have the same pumping rate for bunkering. The reason is that data from the collaborating tramp operator indicates that the majority of their bunker ports do not allow concurrent bunkering, and that we want to be able to differentiate bunker options on price and geographic location rather than 'timing'. This allows us to explore the solution's sensitivity to changes in bunker prices. Furthermore, when taking the time for port clearance, berthing etc. into account, the time spent pumping bunker onboard the ships can be considered negligible. Considering the trade off between solution time and complexity, we therefore assume a fixed time for bunkering at all options regardless of the amount purchased.

7.3.3 Mathematical model

When including bunkering in the problem, bunker stops can be made in between pickup and delivery of a cargo so the two tasks can no longer be aggregated into one. Instead the problem must be modeled similar to a multiple cargo problem. A mathematical arc flow formulation for the pure routing and scheduling problem for multiple cargoes can be found in Christiansen et al. (2007). It is formulated as a pickup and delivery problem with time windows and capacity constraints.

Let $\mathcal{V}$ be the set of ships and index it by v, and let $\mathcal{B}$ denote the set of bunker options indexed by k. Since not all ships are compatible with all ports, we get ship specific bunker sets denoted $\mathcal{B}^v \subseteq \mathcal{B}$. Furthermore, we assume that there are N cargoes and index them by i. Let $\mathcal{N}_P = \{1, \cdots, N\}$ and $\mathcal{N}_D = \{N+1, \cdots, 2N\}$ denote the set of pickup and discharge nodes, respectively. We represent each cargo i by a pickup node $i \in \mathcal{N}_P$ and a discharge node $N+i \in \mathcal{N}_D$. We define $\mathcal{N} = \mathcal{N}_P \cup \mathcal{N}_D$ as the set of all cargo related nodes and partition $\mathcal{N}_P$ into $\mathcal{N}_P = \mathcal{N}_C \cup \mathcal{N}_O$, where $\mathcal{N}_C$ and $\mathcal{N}_O$ contain pickup nodes for contract cargoes and optional cargoes, respectively. Associated with each ship v is now a standard network $(\mathcal{N}^v, \mathcal{A}^v)$ not including bunker options. The standard network nodes, $\mathcal{N}^v \subseteq \mathcal{N}_P \cup \mathcal{N}_D \cup \{o(v), d(v)\}$, correspond to cargoes that ship v is able to carry and two ship specific nodes representing, respectively, the origin and an artificial destination for ship v. Ship v is able to carry a cargo i if it has sufficient capacity, is compatible with the specific load and discharge ports and is in general compatible with cargo i on all accounts. The ship specific cargo nodes are given by $\mathcal{N}_P^v = \mathcal{N}_P \cap \mathcal{N}^v$ for pickups and $\mathcal{N}_D^v = \mathcal{N}_D \cap \mathcal{N}^v$ for discharges. The set of standard network arcs, $\mathcal{A}^v$, is a subset of $\{(i,j) | i \in \mathcal{N}^v, j \in \mathcal{N}^v\}$ and contains all the arcs traversable by ship v, e.g. with respect to time and bunker consumption.

For each ship v we extend the standard node set, $\mathcal{N}^v$, by adding a node for each element in $\mathcal{B}^v$ and index the full set by i. Likewise, we extend the standard arc set, $\mathcal{A}^v$, by adding all arcs connecting nodes in $\mathcal{N}^v \setminus d(v)$ with nodes in $\mathcal{B}^v$ and traversable by ship v with respect to time and bunker. We do not connect the destination node, $d(v)$, with bunker nodes, as we do not want idle ships to bunker since this could send them in the exact opposite direction of their next (unplanned) port stop and since their unplanned voyages could involve port stops with very attractive prices. For each ship v we thereby obtain an extended cargo-bunker network $(\hat{\mathcal{N}}^v, \hat{\mathcal{A}}^v) = (\mathcal{N}^v \cup \mathcal{B}^v, \mathcal{A}^v \cup \mathcal{A}_B^v)$ where $\mathcal{A}_B^v$ denotes arcs connecting bunker nodes to nodes in $\mathcal{N}^v \setminus d(v)$.

For $v \in \mathcal{V}$ and $i \in \hat{\mathcal{N}}^v$ we let l_i^v denote the load onboard ship v just after completing service at node i. In case of full shiploads it is common industry practice to simply distinguish between a laden and a ballast ship, i.e. loaded or empty, rather than calculating the actual load. We will adopt this practice and thereby use binary load variables, l_i^v, equal to 1 if the ship is laden and 0 if the ship is ballast. With $(i,j) \in \hat{\mathcal{A}}^v$ we associate a load dependent time consumption $T_{ij}^v(l_i^v)$ when traversed by ship v calculated from the arrival at the port of node i until the arrival at the port of node j. It accounts for service time at the port of node i whether it is a loading, discharging or bunkering node, and the sailing time from the port of node i to the port of node j. We also associate a load dependent bunker consumption $B_{ij}^v(l_i^v)$ that accounts for bunker consumption while traveling from node i to node j but not including bunker consumption while in port at node i. This port consumption is instead accounted for by B_i^v. Finally, we have the variable cost function $C_{ij}^v(l_i^v)$. Like time consumption, this accounts for costs related to visiting the port of node i and sailing costs from the port of node i to the port of node j. Note however, that the cost of purchasing bunker is not included, as it is a dynamic node cost dependent on the amount of bunker purchased at the node. Instead, this cost will be added separately and accounted at a bunker unit price of P_k for $k \in \mathcal{B}$ while bonus bunker is 'resold' at a unit price P. Also note that if nodes i and j correspond to the same physical port, $C_{ij}^v(l_i^v)$ does not include port costs, $T_{ij}^v(l_i^v)$ does not include travel time and $B_{ij}^v(l_i^v) = 0$. We also have a revenue R_i for all cargoes $i \in \mathcal{N}_P$ and we denote the bunker capacity of ship v by B_{Max}^v. The safety level for bunker inventory is denoted B_{Min}^v while the initial bunker level onboard the ship is denoted B_0^v. Finally, we denote by $[T_{MNi}^v, T_{MXi}^v]$ the time window associated with node $i \in \hat{\mathcal{N}}^v$. For $o(v)$ this window is collapsed into the time ship v is available for service. For the mathematical formulation we need the following variables:

x_{ij}^v, $v \in \mathcal{V}$, $(i,j) \in \hat{\mathcal{A}}^v$. Binary variable that is equal to 1, if ship v visits node i just before node j, and 0 otherwise

t_i^v, $v \in \mathcal{V}$, $i \in \hat{\mathcal{N}}^v$. Denotes the time ship v begins service at node i

l^v_{Bi}, $v \in \mathcal{V}$. Denotes the bunker load onboard ship v just after completing service at node i

l^v_i, $v \in \mathcal{V}$. Binary variable that is equal to 1, if ship v is laden when leaving node i , and 0 otherwise

y^v_k, $v \in \mathcal{V}$, $k \in \mathcal{B}^v$. Gives the bunker purchased by ship v at option $k \in \mathcal{B}^v$

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

$$\max \sum_{v\in\mathcal{V}} \sum_{i\in\mathcal{N}^v_P} R_i\Big(\sum_{j\in\hat{\mathcal{N}}^v} x^v_{ij}\Big) - \sum_{v\in\mathcal{V}} \sum_{(i,j)\in\hat{\mathcal{A}}^v} C^v_{ij}(l^v_i)x^v_{ij} - \sum_{v\in\mathcal{V}} \sum_{k\in\mathcal{B}^v} y^v_k P_k + \sum_{v\in\mathcal{V}} P\cdot(l^v_{Bd(v)} - B^v_0) \tag{7.1}$$

s.t.

$$\sum_{v\in\mathcal{V}} \sum_{j\in\hat{\mathcal{N}}^v} x^v_{ij} = 1, \qquad \forall i \in \mathcal{N}_C, \tag{7.2}$$

$$\sum_{v\in\mathcal{V}} \sum_{j\in\hat{\mathcal{N}}^v} x^v_{ij} \le 1, \qquad \forall i \in \mathcal{N}_O, \tag{7.3}$$

$$\sum_{j\in\mathcal{N}^v_P\cup\mathcal{B}^v\cup\{d(v)\}} x^v_{o(v)j} = 1, \qquad \forall v \in \mathcal{V}, \tag{7.4}$$

$$\sum_{i\in\hat{\mathcal{N}}^v} x^v_{ij} - \sum_{i\in\hat{\mathcal{N}}^v} x^v_{ji} = 0, \qquad \forall v \in \mathcal{V},\ j \in \hat{\mathcal{N}}^v \setminus \{o(v), d(v)\}, \tag{7.5}$$

$$\sum_{i\in\mathcal{N}^v_D\cup\mathcal{B}^v\cup\{o(v)\}} x^v_{id(v)} = 1, \qquad \forall v \in \mathcal{V}, \tag{7.6}$$

$$\sum_{j\in\mathcal{B}^v} x^v_{ij} = 0, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{B}^v, \tag{7.7}$$

$$x^v_{ij}(t^v_i + T^v_{ij}(l^v_i) - t^v_j) \le 0, \qquad \forall v \in \mathcal{V},\ (i,j) \in \hat{\mathcal{A}}^v, \tag{7.8}$$

$$T^v_{MNi} \le t^v_i \le T^v_{MXi}, \qquad \forall v \in \mathcal{V}, i \in \hat{\mathcal{N}}^v, \tag{7.9}$$

$$x^v_{ij}(l^v_i + 1 - l^v_j) = 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \hat{\mathcal{A}}^v | j \in \mathcal{N}^v_P, \tag{7.10}$$

$$x^v_{i,N+j}(l^v_i - 1 - l^v_{N+j}) = 0, \qquad \forall v \in \mathcal{V}, (i,N+j) \in \hat{\mathcal{A}}^v | j \in \mathcal{N}^v_P, \tag{7.11}$$

$$x^v_{ij}(l^v_i - l^v_j) = 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \hat{\mathcal{A}}^v | j \in \mathcal{B}^v, \tag{7.12}$$

$$l^v_{o(v)} = 0, \qquad \forall v \in \mathcal{V}, \tag{7.13}$$

$$l^v_i = 1, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{N}^v_P, \tag{7.14}$$

$$l^v_{N+i} = 0, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{N}^v_P, \tag{7.15}$$

$$x^v_{ij}(l^v_{Bi} - B^v_{ij}(l^v_i) - B^v_j + y^v_j - l^v_{Bj}) = 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \hat{\mathcal{A}}^v | j \in \mathcal{B}^v, \tag{7.16}$$

$$x^v_{ij}(l^v_{Bi} - B^v_{ij}(l^v_i) - B^v_j - l^v_{Bj}) = 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \hat{\mathcal{A}}^v | j \in \mathcal{N}^v, \tag{7.17}$$

$$l^v_{Bo(v)} = B^v_0, \qquad \forall v \in \mathcal{V}, \tag{7.18}$$

$$B^v_{\text{Min}} + \sum_{j\in\hat{\mathcal{N}}^v} B^v_{ij}(l^v_i)x^v_{ij} \le l^v_{Bi} \le \sum_{j\in\hat{\mathcal{N}}^v} B^v_{\text{Max}} x^v_{ij}, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{B}^v \cup \mathcal{N}^v \tag{7.19}$$

$$0 \le y^v_i \le (B^v_{\text{Max}} - B^v_{\text{Min}}) \sum_{j\in\hat{\mathcal{N}}^v} x^v_{ij}, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{B}^v \tag{7.20}$$

$$t^v_i + T^v_{i,N+i}(l^v_i) - t^v_{N+i} \le 0, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{N}^v_P, \tag{7.21}$$

$$\sum_{j\in\hat{\mathcal{N}}^v} x^v_{ij} - \sum_{j\in\hat{\mathcal{N}}^v} x^v_{j,N+i} = 0, \qquad \forall v \in \mathcal{V},\ i \in \mathcal{N}^v_P, \tag{7.22}$$

$$l^v_i \in \{0,1\}, \qquad \forall v \in \mathcal{V},\ i \in \hat{\mathcal{N}}^v, \tag{7.23}$$

$$x^v_{ij} \in \{0,1\}, \qquad \forall v \in \mathcal{V},\ (i,j) \in \hat{\mathcal{A}}^v. \tag{7.24}$$

The objective function (7.1) maximizes profit by subtracting all costs from revenues for serviced cargoes and adding the value of bonus bunker. Note that B_0 is a constant that could be removed. However, we leave it in the objective function to emphasize that only bunker levels above the initial level incurs a premium. The premium for bonus bunker is negative if the bunker level at the destination node is less than at the origin node. Constraints (7.2) and (7.3) ensure that all contract cargoes are carried by exactly one ship and that all spot cargoes are carried by at most one ship. Constraints (7.4) and (7.6) together with the flow conservation constraints in (7.5) ensure that each ship is assigned a schedule starting at the origin node and ending at the destination node. If a ship is idle for the entire planning period, the assigned schedule is simply represented by the arc $(o(v), d(v))$. Our assumption of at most one bunker stop in between cargo stops is enforced in constraints (7.7). In situations where this assumption is not reasonable, the right hand side of the constraints can be increased or they can just be removed from the model. Constraints (7.8) ensure that if the route for a ship v visits node i directly before node j, the service at node j cannot begin before service time at node i plus the service time at node i and travel time from node i to node j with ship v. Waiting time is allowed and, hence, the constraints have an inequality sign. Together with the time window constraints (7.9) they take care of the temporal aspect of the problem. If ship v does not visit node i, the service time t_i^v is artificial. Constraints (7.10), (7.11) and (7.12) ensure that the cargo load variables are correctly updated along the chosen route, increasing the load variable by 1 if visiting a loading node, similarly decreasing the load variable if visiting a discharge node and simply maintaining the previous load variable value if visiting a bunker node. In (7.13) the initial load condition for each ship is given since we assume that the ship is empty at the time it is available for service. Constraints (7.14) and (7.15) determine the value of the load variables for loading and discharging nodes, respectively. For completeness constraints (7.15) have been added to the model though they could be omitted. Constraints (7.16)-(7.19) place similar restrictions on the bunker load variables: Constraints (7.16) and (7.17) ensure that the bunker load variables are updated correctly, constraints (7.18) give the initial bunker level for each ship while (7.19) give lower and upper bounds for the variables ensuring that a ship will never arrive at a port with less bunker than the safety level and will never carry more bunker than its bunker capacity allows. Constraints (7.20) restrict the bunker purchase amounts for each ship. Constraints (7.21) are precedence constraints ensuring that a cargo cannot be discharged before it has been picked up, i.e. node i must be visited before node $N + i$. Constraints (7.22) couple pickup and discharge nodes for each cargo together to ensure that the same ship will service both nodes. Finally, the load variables and the flow variables are restricted to be binary in (7.23) and (7.24), respectively.

7.4 Solution Method

The mixed integer programming model (7.1)-(7.24) could in theory be solved by commercial optimization software for non-linear problems. In practice, however, 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 TSRSPBO.

In the mathematical programming model (7.1)-(7.24), constraints (7.4)-(7.24) are ship specific with no interaction between ships. They constitute a routing and scheduling problem for each ship where time windows, cargo and bunker capacity as well as bunker purchases are considered. We denote these ship specific constraints *ship routing constraints* and further notice that the objective function also splits into separate terms for each ship. The only constraints linking the ships together are the so called *common constraints* in (7.2) and (7.3) which ensure that each contract cargo is carried by exactly one ship and that each spot cargo is carried by at most one ship. 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 remain in the *master problem* in which feasible ship schedules constitute the columns. This way the original problem is transformed into a master problem with a reduced number of constraints but with a potentially very large number of columns.

Often ship scheduling problems are so tightly constrained that it is possible to a priori generate all master problem columns. This is done by generating the optimal schedule for each feasible cargo set for each ship. Such an approach has been attempted in Brønmo et al. (2007). However,

as already mentioned, Brønmo et al. (2010) find it computationally advantageous to apply dynamic column generation even though a priori generation can be applied. The inclusion of bunker decisions in the scheduling process will further complicate the determination of an optimal schedule for a given cargo set and can, hence, make a priori generation very time consuming. In line with this, we apply dynamic column generation to solve the problem (see e.g. Desaulniers et al. (2005) for a general description or Christiansen et al. (2007) for a maritime version). Therefore, we initially consider only a subset of the master problem columns and iteratively add new columns that have the potential to improve the current solution. We find these columns by iteratively solving the subproblems, also called pricing problems.

7.4.1 The Master Problem

The common constraints (7.2) and (7.3) in combination with the objective function (7.1) 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 $\mathcal{R}^v$ denote the set of all feasible schedules for ship v. Each cargo set can correspond to several feasible schedules as the order of cargoes in the schedule will correspond to different geographical routes. Schedules can also differ in bunker port calls, the amounts purchased at each bunker port and even in the timing of port calls. For a given set of cargoes there will be at least one profit maximizing schedule corresponding to the optimal bunkering strategy and the optimal timing of port calls. However, due to the subproblem solution method we might generate several different schedules for the same cargo set. We denote the profit of a schedule by p_r^v for $r \in \mathcal{R}^v$ and define a binary schedule variable λ_r^v that is equal to 1 if ship v is chosen to sail schedule r, and 0 otherwise. The profit p_r^v is calculated based on information from the underlying schedule, which holds all necessary information, i.e. the ship it is constructed for, the cargoes carried, the bunker ports visited as well as the bunker quantities purchased, and the timing of port calls during the schedule. Finally, we let a_{ir}^v be equal to 1 if ship v carries cargo i in schedule r, and 0 otherwise.

The master problem is now given by the following path flow reformulation of the original arc flow model:

$$\max \sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}^v} p_r^v \lambda_r^v \tag{7.25}$$

s.t.

$$\sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}^v} a_{ir}^v \lambda_r^v = 1, \qquad \forall i \in \mathcal{N}_C, \tag{7.26}$$

$$\sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{R}^v} a_{ir}^v \lambda_r^v \leq 1, \qquad \forall i \in \mathcal{N}_O, \tag{7.27}$$

$$\sum_{r \in \mathcal{R}^v} \lambda_r^v = 1, \qquad \forall v \in \mathcal{V}, \tag{7.28}$$

$$\lambda_r^v \in \{0,1\}, \qquad \forall v \in \mathcal{V},\ r \in \mathcal{R}^v. \tag{7.29}$$

The above model is based on all feasible schedules but it is not necessary to include all of them. Instead, column generation is applied to dynamically generate them as needed. This process begins with the solution of the *restricted master problem* (RMP) which is the linear relaxation of the original master problem (7.25)-(7.29) but with only a subset of the columns included. Iteratively we then generate new promising columns by solving the subproblems.

7.4.2 The Subproblem - Generation of promising schedules

Constraints (7.4)-(7.24) split into one independent subproblem for each ship. Since these are all essentially the same problem, we simply consider the generic subproblem for ship v and refer to 'the subproblem'. Note though the interdependence between the subproblems due to the common constraints. The ship routing constraints in the subproblem ensure that any solution is a feasible schedule for ship v and the objective ensures that only schedules with the potential to improve the current solution of the RMP are generated. This, in turn, means finding schedules with positive

reduced costs in the current solution of the RMP, i.e. finding columns r with $p_r - \sigma^T a_r$, where σ is the dual vector of the current solution of the RMP. Let u_i be the dual variables for constraints (7.26) and (7.27) where the variables corresponding to (7.26) are free of sign while the variables corresponding to (7.27) must be nonnegative. Next, let w_v be the dual for constraint (7.28) which is also free of sign. Finally, define $\sigma_i = u_i$ for all $i \in \mathcal{N}_P$, $\sigma_{o(v)} = w_v$ corresponding to the origin node and $\sigma_i = 0$ for all other i. Since we consider the generic subproblem we can drop the superscript v and the subproblem is then given by:

$$\max \sum_{i \in \mathcal{N}_P} R_i\Big(\sum_{j \in \hat{\mathcal{N}}} x_{ij}\Big) - \sum_{(i,j) \in \hat{\mathcal{A}}} (C_{ij}(l_i) + \sigma_i) x_{ij} - \sum_{k \in \mathcal{B}} y_k P_k + P(l_{Bd} - B_0) \tag{7.30}$$

s.t.

$$(7.4) - (7.24). \tag{7.31}$$

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 will be represented by a new column in the RMP. The subproblem can be modeled as a resource constrained shortest path problem and is $\mathcal{NP}$-hard since it is a generalization of the shortest path problem with time windows which is itself $\mathcal{NP}$-hard (see e.g. Desrosiers et al. (1995)). We therefore devote Section 7.4.4 to an efficient solution method for the subproblem.

7.4.3 Full Column Generation Scheme

The full column generation scheme is an iterative process starting from the RMP with only a small initial column set. To ensure feasibility of the initial problem, we include a dummy column for each contract cargo. Each dummy column corresponds to an artificial ship carrying exactly one contract cargo. The revenue from each dummy schedule is $-M$, where M is a large constant. Feasibility with respect to the generalized upper bound constraints (7.28) must also be ensured, i.e. each ship must be assigned a schedule. Therefore, we include an empty schedule with 0 profit for each ship corresponding to the ship being idle for the entire planning horizon, and leave the corresponding $(o(v), d(v))$ out of the subproblem networks.

Once the RMP has been solved, the optimal dual solution values are transferred to each of the subproblems which are then solved to obtain new schedules. The subproblem solution method presented in the next section allows us to speed up the solution process considerably by generating several columns for each ship in each iteration. This is done by transforming all new schedules with positive reduced cost into columns rather than just transforming the schedule with the maximum reduced cost. All these new columns are then added to the RMP which is resolved to obtain new dual values that can again be transferred to the subproblems. This process of iterating between the master problem and the subproblems continues until no promising columns can be found, i.e. until no schedules have positive reduced costs.

Since all the intricate and nonlinear constraints and costs are transferred to the subproblems, the master problem can most often be solved by commercial linear programming software. As each subproblem must be solved a potentially great number of times to obtain all necessary columns, a fast solution method for the subproblem is vital for the effectiveness of the column generation scheme. Since the subproblem considered here is $\mathcal{NP}$-hard, solving it can be very time consuming and a choice between heuristics and optimization must be made depending on the desired solution quality and computation time.

Once the column generation process terminates, an optimal solution to the linear relaxation of the full master problem is obtained. In order to ensure an optimal integral solution, the column generation scheme must be embedded in a Branch & Bound search, resulting in a Branch & Price algorithm. In our computational studies we have, however, encountered relatively few fractional solutions and for these fractional solutions, the integrality gap was acceptable. Therefore, we have not implemented a Branch & Bound algorithm. Instead, integrality has been enforced by the simple, but non-optimal, approach of solving the integer version of the RMP once column generation terminated. In Section 7.7 we verify the quality of these forced integer solutions by comparing them to their corresponding upper bounds obtained from the fractional solutions.

7.4.4 Solving the subproblem

Shortest path problems with resource constraints (SPPRC) are often encountered in both land and air based transportation, e.g. as a subproblem in column generation frameworks for solving vehicle routing and crew rostering problems. The problem entails finding a shortest path between two nodes, while satisfying several resource constraints. For vehicle routing problems typical resources include, among others, time windows and vehicle capacity, and we observe similar requirements here. The SPPRC is typically solved by dynamic programming algorithms on the underlying networks, and we will also use this approach here. The reader is referred to Desaulniers et al. (1998), Irnich and Desaulniers (2005) and Irnich (2008) for a thorough introduction to the SPPRC, the related dynamic programming algorithms, and several associated concepts. In what follows we provide more specific details on the SPPRC at hand. In particular, we discuss in detail the nature of the underlying network, preprocessing techniques, and the actual dynamic programming algorithm we implement.

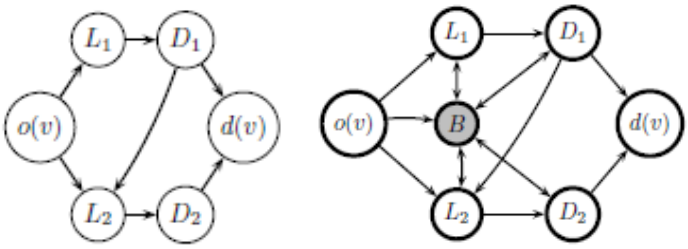
The Underlying Ship Network

Before providing a detailed description of the underlying network for the subproblem, we state an assumption which impacts the modelling approach chosen. As the research in this paper is more on a tactical level than an operational one, we are looking for a guide line on where to bunker and roughly how much to bunker at each bunker stop. The operational planning problem of exactly how many tons of bunker to purchase with decimal accuracy is not relevant in our setting where decisions are based on bunker price forecasts rather than actual prices. If decimal precision is desired for the bunker levels, one could adopt similar procedures from Ioachim et al. (1998) and Christiansen and Nygreen (2005) to handle the linear node costs. Instead, we assume that we can discretize the bunker purchase variables; this allows us to avoid the difficulties associated with linear node costs.

When discretizing the bunker purchase variables an obvious concern is how to do this in a manner that does not sacrifice optimality too much. We therefore note that logically the optimal decision at each bunker stop is to either fill up the tank or to purchase just enough bunker to allow the ship to sail to the next bunker stop; we cannot know in advance how much bunker is required to get to the next bunker stop, but we can obviously fill up the tank. When tuning and testing the devised solution method in Section 7.6 and Section 7.7 we see that when constructing bunker schedules, the majority of bunker stops actually correspond to filling up the tank to its capacity. As we want to retain this optimal decision amongst the possible purchase quantities, we let each bunker node correspond to the situation of filling up the tank to a certain inventory level rather than purchasing a specific amount of bunker. Note that using a mix of these two types of bunker purchases can almost aggregate some bunker nodes, e.g. if the 'fill up tank' node corresponds to bunkering 833 tons and the fixed amount node is 800 tons. Therefore, we only use 'fill up to' nodes so that the purchase quantities span the feasible interval of possible purchase quantities as much as possible. That is each bunker node of the network corresponds to a different 'fill up to' level for a particular bunker purchase option. Each ship has its own bunker tank capacity and safety level and these, in turn, describe the ship specific interval of feasible bunker purchase quantities. This interval is divided in to L discrete bunker purchase quantities, where L is a parameter of the algorithm that we tune in Section 7.6. Note that this parameter could be different for each ship but as the tank capacities do not vary too much in size for the fleet we consider, we have chosen to use the same parameter for all ships. Dividing $(B_{Max} - B_{Min})$ into L intervals and rounding the result down to the nearest 25 tons yields the refinement level q. That is each bunker option $i \in \mathcal{B}^v$ is replaced by the L discrete purchase quantity levels: B_{Max}, $B_{Max} - q \ldots$, $B_{Max} - (L-1)q$. If $L = 1$, the only option is to fill up the tank to its maximum capacity. Note that with the discretization of bunker purchases, the notion of bonus bunker becomes a vital part of the solution procedure. If this premium for unused bunker is not included in the model, it becomes more important to find combinations of bunker purchase quantities that let ships finish their schedules with empty tanks than to find cheap bunker options. In other words, the effect of price changes on the optimal bunker plan is almost non existent.

The SPPRC for ship $v \in \mathcal{V}$ can now be stated on a network in which the node set is comprised of $\mathcal{N}^v$ and an extended set of bunker nodes based on the bunker set $\mathcal{B}^v$. This extended set of

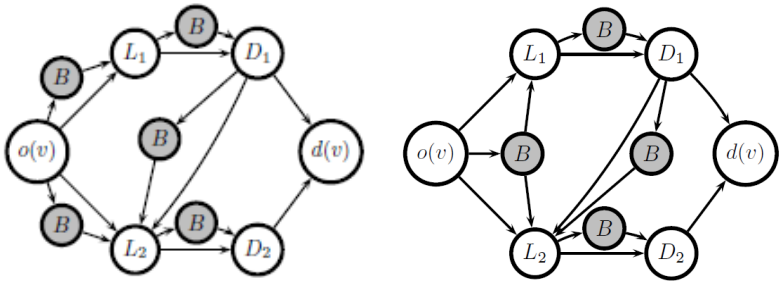
bunker nodes is obtained by discretizing the bunker purchase variables in to the L discrete bunker purchase options. Arcs are introduced to govern the transitions between the cargo nodes and the bunkering possibilities. More specifically, the arc set contains all arcs in $\mathcal{A}^v$ (i.e. the cargo network), and additional arcs which connect cargo nodes to each bunkering possibility. Figures 7.1(a) and 7.1(b) illustrate the resulting network for a small example with 2 cargoes and only one bunker option that has been discretized into just one node, i.e. $L = 1$. Note the double headed arrows in Figure 7.1(b) which have just been aggregated for the sake of simplicity in this figure. Nodes L_i and D_i correspond, respectively, to pickup and discharge of cargo i while node B corresponds to the bunker node.



(a) Standard cargo network (b) Normal cargo-bunker network

Figure 7.1: Extending a standard cargo network to include bunker nodes

While Figure 7.1(b) can be used to model the SPPRC for ship $v \in \mathcal{V}$, we prefer to modify it in the following way. We duplicate the extended set of bunker nodes, a set with cardinality $K \cdot L$ (with $K = |\mathcal{B}^v|$), as many times as there are potential time feasible arcs in the standard network, $(\mathcal{N}^v, \mathcal{A}^v)$, excluding arcs leading to the destination node. Note that 'potential time feasible' means that any arc connecting nodes in $\mathcal{N}^v \setminus d(v)$, and which is feasible with respect to time, is considered here. A 'potential time feasible' arc does not have to belong to $\mathcal{A}^v$ as it can be infeasible with respect to bunker constraints. Each new node is inserted on to the edge it is associated with. If any arcs connecting bunker nodes to the rest of the network are infeasible with respect to bunker consumption and safety levels, they can be removed. Essentially, we construct multiple nodes in the network for each $i \in \mathcal{B}^v$, where each node is associated with a specific, discrete bunker amount and indicates whether the respective bunkering occurs between a specific origin destination pair. The resulting, extended network can be seen in Figure 7.2(a).



(a) Extended cargo-bunker network (b) Reduced network

Figure 7.2: The extended bunker network and the reduced extended cargo bunker-network

Constructing such networks means that all paths in the networks respect the cargo capacity and precedence constraints as they are implicitly considered in the network setup. Furthermore, each arc in the network now corresponds to either a laden or a ballast ship and, hence, the load dependent time and bunker consumption as well as costs can now be considered fixed and calculated in advance. As a result, in total, this network setup allows us to disregard the onboard cargo as a resource, and this simplifies the dynamic programming algorithm for finding the shortest path.

All these advantages do, however, come at the expense of an increase in the potential node and arc count of $(N^2 + N - 1)K \cdot L$ and $(2N^2 - 2N - 1)K \cdot L$, respectively, compared to the normal cargo-bunker network illustrated in Figure 7.1(b). A significant number of these extra nodes and arcs can, however, be removed by preprocessing, especially with respect to time windows. A further reduction in the node and arc count can be achieved by noting that some of these new bunker nodes can be aggregated into one without sacrificing the reduction in resources. First, all bunker nodes that are successors to the origin node and correspond to the same bunker option and purchase level can be aggregated. Secondly, all bunker nodes that are successors to a single discharge node and predecessors to other pickup nodes and correspond to the same bunker option and purchase level can be aggregated. This reduces the potential size of the network by $(N^2 - N - 1)K \cdot L$ nodes and also $(N^2 - N - 1)K \cdot L$ arcs compared to the extended cargo-bunker network illustrated in Figure 7.2(a). This gives a potential network size of $2N + 2 + (2N + 1)K \cdot L$ nodes and $N^2 + 2N + (N^2 + 3N + 1)K \cdot L$ arcs. Figure 7.2(b) illustrates the final network for the same example above.

Time windows on bunker nodes are, as mentioned, so narrow that a ship can never visit the same bunker node twice. In our data sets, the time windows for cargo loading are also tight enough that, in combination with the long voyage lengths, there do not exist any time feasible cycles in the networks. Hence, the resulting network is acyclic. If cycles do exist, nodes with wide time windows must be split in to several duplicate nodes with smaller time windows. This produces an acyclic network; however, it does not ensure that cargoes are not lifted several times in one schedule as this simply corresponds to visiting several of the duplicate nodes for the same cargo. To prevent this, the dynamic programming algorithm needs to remember all previously visited nodes. This is cumbersome and we refrain from doing so. We describe how this is handled in the following section.

During network construction, standard preprocessing techniques are applied to tighten time windows and, in turn, reduce the number of arcs, see e.g. Desrosiers et al. (1995). Each node has an associated time window, an associated port and for bunker nodes also a bunker price. Furthermore, each node also has a bunker window holding the minimum and maximum level of bunker allowed onboard a ship on arrival. For cargo nodes and the destination node this window is $[B^v_{Min}, B^v_{Max}]$ and for the origin node it is $[B^v_0, B^v_0]$. For the bunker nodes, the window becomes $[B^v_{Min}, F]$, where F is the 'fill up to' level at the corresponding bunker node. Each arc in the network now has a constant time and bunker consumption as well as cost and we denote these by T_{ij}, B_{ij} and C_{ij}, respectively. We abuse notation slightly by now letting B_{ij} denote bunker consumption corresponding to both traveling from i to j as well as the consumption from port operations at node i (as opposed to $B^v_{ij}(l^v_i)$ that did not include port operations at node i). Note that C_{ij} still does not include bunker purchases or the bonus bunker premium as these will be dynamically added.

Dynamic Programming Algorithm

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

$$\hat{C}_{ij} = C_{ij} + \sigma_i - R_i \qquad \forall (i,j) \in \hat{\mathcal{A}}, \tag{7.32}$$

where $R_i = 0$ for $i \notin \mathcal{N}_P$. The remaining part of the reduced cost expression in (7.30), i.e. the node costs for bunker purchases and node premiums for bonus bunker, must be added dynamically as partial schedules are extended and bunker purchase amounts are determined.

As mentioned above, we solve the SPPRC using a dynamic programming algorithm. Such algorithms for this particular problem build new schedules for ship $v \in \mathcal{V}$ by starting with the trivial, partial schedule $s = \{o(v)\}$. 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_i ending in node i we associate a label $\mathcal{L}(s_i) = (\bar{C}(s_i), T(s_i), B(s_i))$. Here $\bar{C}(s_i)$ is the negative of the reduced cost for the schedule, i.e. the sum of the arc and node costs where the

arc costs are $\sum_{(i,j)\in s_i} \hat{C}_{ij}$. $T(s_i)$ and $B(s_i)$ denote, respectively, the arrival time at node i and the bunker inventory level on arrival at node i on schedule s_i. Note that we relax the subproblem to allow non elementary paths and refrain from keeping track of the nodes previously visited. Hence, routes can be produced where a cargo is picked up more than once, i.e. have $a_{ir} > 1$ for some i. Such a schedule is not added to the master problem.

Two partial schedules generated at the same node can be compared by defining a partial order relation between the respective labels. This partial order allows us to determine if one label dominates another that can, hence, be discarded. This dominance concept ensures that only the best schedules, i.e. Pareto optimal, are kept during the iterative process of the algorithm as only they can contribute to the optimal schedule. The success of the algorithm relies on an efficient domination procedure for the labels to eliminate non-useful partial schedules. In our case we note that a schedule s_i ending at node i dominates another schedule s'_i also ending at node i if and only if $\mathcal{L}(s_i) \neq \mathcal{L}(s'_i)$, $\bar{C}(s_i) \leq \bar{C}(s'_i)$, $T(s_i) \leq T(s'_i)$ (since there is no cost for waiting) and $B(s_i) \geq B(s'_i)$.

In order to augment a partial path, label extension is necessary. Label extension is associated with a particular arc in the underlying network and utilizes specific resource extension functions that dictate how each resource level will change when traversing the arc. For the case at hand, the time resource is extended using $T(s_j) = \max\{T_{MNj}, T(s_i) + T_{ij}\}$, and this extension is deemed feasible if $T(s_i) + T_{ij} \leq T_{Mxj}$. When extending to bunker nodes, the bunker resource is updated using $B(s_j) = B_{MXj}$ with an associated dynamically calculated purchase quantity $y_j = B_{MXj} - (B(s_i) - B_{ij})$. Note that this updates the bunker inventory immediately on arrival at the node and remember that B_{MXj} is defined as the 'fill up to' level for bunker nodes. I.e. a ship filling up its bunker tank to e.g. 1200 tons might actually leave the node with 1199 tons as a small amount of bunker is consumed while bunkering. If j is a cargo related node, we have $B(s_j) = B(s_i) - B_{ij}$. For the destination node, d, we get $B(s_d) = B_0$, with an associated dynamically calculated bonus bunker amount, y_d, given by $B(s_i) - B_{id} - B_0$. With this setup, s_i can only be extended to node j if $B(s_i) - B_{ij} \geq B_{MNj}$ and $B(s_i) - B_{ij} \leq B_{MXj}$. The latter requirement ensures that a schedule with arrival bunker inventory higher than the 'fill up to' level at a bunker node will not visit such a node. Finally, the negative of the reduced costs are updated as follows:

$$\bar{C}(s_j) = \bar{C}(s_i) + \hat{C}_{ij} + y_j P_j, \qquad \forall j \in \mathcal{B}, \tag{7.33}$$

$$\bar{C}(s_j) = \bar{C}(s_i) + \hat{C}_{ij}, \qquad \forall j \in \mathcal{N}, \tag{7.34}$$

$$\bar{C}(s_d) = \bar{C}(s_i) + \hat{C}_{id} - y_d \cdot P, \tag{7.35}$$

The dynamic programming algorithm we implement is hence a standard label setting algorithm, which begins at $o(v)$ 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. We add all schedules with positive reduced cost, i.e. $\bar{C}(p) < 0$, to the master problem. See Algorithm 1 for a general overview of our label setting algorithm. Due to the reselling of bonus bunker, all schedules will have the same amount of bunker at the end, namely the initial inventory level of the ship. Schedules can, however, differ in both reduced costs and time. Therefore, we can have multiple columns corresponding to the same cargo set in the master problem if they correspond to different end times, e.g. due to differences in bunker plans.

Finally, it should be noted that due to the discretization of bunker purchases and the assumption of at most one bunkering in between cargo stops, the subproblem solution method described above is heuristic. To ensure an optimal solution to the master problem, the subproblems should be solved to optimality once the heuristic approach fails to find schedules with positive reduced costs. As previously discussed, the planning problem considered here is of a more tactical nature and, hence, an optimal continuous solution is beyond the scope of this research. We do, however, rerun the dynamic programming algorithm with an increased number of possible purchase quantities, i.e. an increased value of L, for the fixed cargo routes found by the initial optimization of the master problem. We discuss this further when tuning the algorithm in Section 7.6.

Algorithm 1: Label Setting Algorithm

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

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

7.5 Problem Instance Generators

In order to both tune and test the devised algorithm thoroughly, we have developed instance generators that independently generates cargoes and bunker prices. These instance generators are based on industry data from the collaborating tramp operator. Although this operator operates world wide, the cargoes naturally divide into two groups traveling within two separate parts of the world with only 5% of cargoes traveling between them. We therefore limit our analysis to one of these cargo groups, namely the one responsible for almost 70% of the overall cargoes. We have excluded some remote regions that generate very little, if any, demand. This leaves us with a cargo area covering the Mediterranean, the North-West part of Europe, the East Coast of Canada and the US, the Mexican Gulf and the Caribbean Sea. We have selected 38 ports that are representative for the ports in this area. Both generators therefore assume 38 ports and each port has some associated ship dependent port costs. For all problem instances the fleet is the same and consists of 7 ships of varying size and other characteristics, e.g. speed, bunker consumption etc.

For each cargo, the cargo generator randomly selects a pickup port from a probability distribution of cargo pickup ports. Once the pickup port is known, there is a specific discharge distribution related to this pickup port from which a discharge port is randomly drawn. A cargo quantity is randomly selected in a user defined interval. For our analysis, the quantities are randomly chosen between 60-90% of ship sizes. Based on this quantity as well as the distance between pickup port and discharge port and their costs, a reasonable revenue for transporting the cargo is randomly calculated. Also based on user defined intervals, time windows for both pickup and discharge as well as the service time for loading and unloading are randomly calculated. We have used time windows with a length of minimum 72 hours and maximum 120 hours. Finally, the cargo is randomly selected to be either a spot cargo or a contract cargo depending on user input.

The bunker price generator randomly generates a price quote for each of the 38 ports for a number of consecutive time periods determined by the user. E.g. if a period is determined to be 3 days and the user asks for 20 bunker options, 20 price quotes will be generated for each of the 38 ports and each of these prices are given a time window of 3 days. For the 38 prices of the last time period in the planning period, an average is calculated to use for bonus bunker. To generate the actual prices, the 38 ports are divided into regions and each port is randomly selected to belong to one of the price classes cheap, average and expensive. For each region, reasonable bunker price intervals corresponding to a cheap port, an average port and an expensive port at the beginning of the planning period are given as parameters to the generator. Each port is assigned a start price, e.g. a price for the first 3 days, by randomly picking a price in the interval that corresponds to

the specific region and price class of the port. In order to generate prices for the remaining time periods a world trend is randomly generated that is valid for all regions and ports. This world trend simply defines whether the price goes up or down from one time period to the next. For each port the remaining bunker prices are now determined by using the start price of the port and then raising or lowering the price from time period to time period following the world trend. The actual amount it is raised or lowered with is determined randomly for each port for each time period from an interval defined by user input. In our analysis we have used an interval of 0-5%.

7.6 Parameter Tuning

The number of possible purchase levels, i.e. the parameter L, must be tuned before running the algorithm. Obviously, the more levels we allow the more of the underlying feasible bunker interval we span, however at a cost of computation time.

We have generated 18 problem instances for tuning using the instance generators described in Section 7.5. All problem instances have the same fleet of 7 ships and use 38 ports worldwide. Three cargo instances have been generated containing 30 cargoes with their loading time windows distributed over a time horizon of 30 days. Note that the planning period continues after these 30 days as cargoes must of course also be discharged. Three bunker price instances have been generated with 14 weekly bunker options for each port corresponding to a time horizon that just contains the latest possible discharge time plus the time to discharge. Combining each cargo instance with each of the corresponding bunker instances yields nine instances with this combination of data and we denote them C30/PH30/B14 instances. Another three cargo instances have been generated with 50 cargoes over a 60 days pickup planning horizon. Three bunker price instances have been generated for these cargo sets but now with 19 bunker options for each port. Again, we get nine instances by combining each cargo instance with each of the bunker instances. We denote these bigger instances by C50/PH60/B19.

The integrated planning approach will be more advantageous the fewer contract cargoes there are. We want to explore what effect the integrated planning approach can at best have and therefore we have predefined all cargoes to be spot cargoes. However, when testing the algorithm in Section 7.7 we also consider data sets with contract cargoes.

On each of the problem instances we have run the algorithm with varying number of purchase levels, namely L varying from one to ten, and report the key values in Table 7.1. Each entry corresponds to the average over the nine problem instances of the specific instance type for the stated setting of L. The key values reported are: the percentage increase in the objective function value compared to the $L = 1$ case (Obj.), the total running time in CPU seconds ($\text{CPU}_{\text{Total}}$), the CPU seconds for solving the subproblems (CPU_{Sub}), and, finally, the percentage of all bunker stops that filled up the ship's bunker tank to its maximum capacity (Filled). This number is interesting as it shows that relatively few bunker stops use 'fill up to' levels lower than tank capacity and, hence, the actual discretization of this interval is less important.
We see from Table 7.1 that increasing L yields an objective function value increase of only 0.53-1.4% and that the increase is largest when going from $L = 1$ to $L = 2$. Note also that the objective function value does in one case drop when increasing L. This demonstrates the heuristic nature of the algorithm due to the discretization. Furthermore, the increase in computation time is considerable as L is increased and this is almost only due to the increase in solution time for the subproblems. Running the algorithm with high L values is therefore computationally undesirable. Increasing L gradually during the algorithm as the optimum is approached will also be very time consuming. Even resolving the subproblems in each iteration with an increased value of L for each fixed cargo route found by the shortest path solver, i.e. each Pareto optimal schedule (or the best of them), will be computationally expensive. Instead we have chosen to investigate the effect of simply increasing L for the fixed cargo sets found for each ship in the final solution to the master problem. We rerun the algorithm with $L = 17$ as this is the lowest value that yields a refinement of 25-50 mts between purchase quantities for all ships. For our tactical approach this level of refinement is sufficient and mimics a continuous solve.

Table 7.2 shows the key values for rerunning the algorithm on all 18 instances again for increasing values of L but this time finishing the algorithm by solving the bunker optimization problem

Table 7.1: Tuning results for increasing L values

	C30/PH30/B14				C50/PH60/B19			
L	Obj.	CPU_{Total}	CPU_{Sub}	Filled	Obj.	CPU_{Total}	CPU_{Sub}	Filled
1	-	5.8	5.5	100.0	-	31.6	30.9	100.0
2	0.91	14.0	13.6	78.6	0.53	77.2	76.1	77.3
3	1.08	23.9	23.4	74.4	0.58	116.5	115.1	78.8
4	1.18	35.2	34.6	73.1	0.69	186.9	185.2	72.4
5	1.25	48.5	47.8	75.2	0.74	276.9	274.7	70.2
6	1.32	71.7	70.9	74.4	0.81	360.6	358.0	67.9
7	1.34	86.7	85.7	73.1	0.86	495.5	492.5	67.3
8	1.35	111.1	110.0	72.5	0.88	540.9	537.7	64.6
9	1.37	144.4	143.2	72.5	0.86	755.0	751.4	64.9
10	1.40	157.2	155.9	73.1	0.89	861.5	857.6	64.3

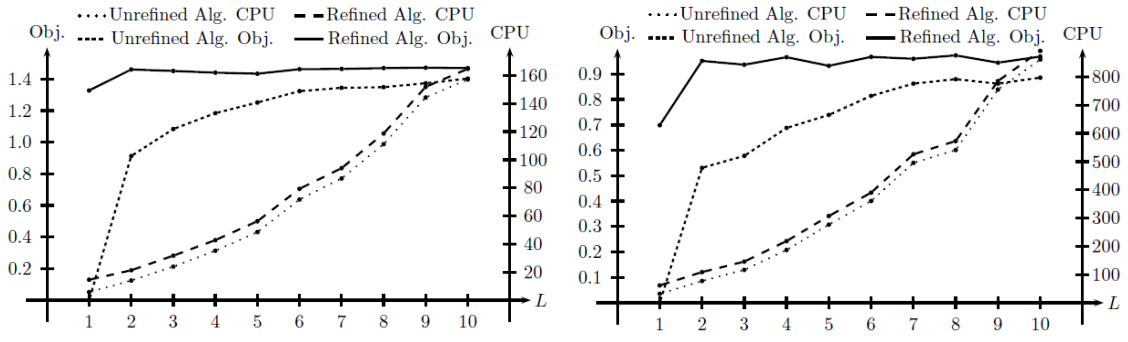
with $L = 17$ for the fixed cargo sets determined by the final solution to the master problem. If the original solution from using the low L-value is better than when rerunning the algorithm with $L = 17$ for fixed routes, we naturally use the original solution rather than the solution from rerunning with $L = 17$. Note that the objective column (Obj.) again contains the percentage increase in profit compared to the $L = 1$ case for the unrefined algorithm, i.e. the one used in Table 7.1.

Table 7.2: Tuning results for algorithm that reoptimizes with $L = 17$ for increasing L values

	9 instances: C30/PH30/B14			9 instances: C50/PH60/B19		
L	Obj.	CPU_{Total}	Filled	Obj.	CPU_{Total}	Filled
1	1.33	14.6	76.6	0.70	61.5	66.0
2	1.46	21.3	73.8	0.95	108.7	65.5
3	1.45	31.7	73.5	0.94	145.8	65.2
4	1.44	42.8	73.6	0.97	218.6	64.4
5	1.43	56.2	74.4	0.93	307.3	65.7
6	1.46	79.3	72.5	0.97	390.2	63.5
7	1.46	94.0	75.0	0.96	526.0	64.5
8	1.47	118.7	73.8	0.97	572.8	64.3
9	1.47	151.9	73.3	0.94	785.3	63.6
10	1.47	164.8	73.8	0.97	892.1	64.1

As before, we see that increasing the value of L yields almost no, if any, increase in profit and yet the computation time increases rapidly.

In Figure 7.3(a) and Figure 7.3(b) we illustrate these findings for the C30/PH30/B14 and the C50/PH60/B19 instances, respectively. Each figure shows a plot of the percentage increase in objective function value and the CPU seconds both as functions of L for the standard version of the algorithm and for the refined version using a resolve on fixed cargo sets with $L = 17$. As can be seen from the above figures, the refined algorithm with $L = 2$ afterwards increased to $L = 17$ yields almost the best objective function values of all settings and at almost no increase in computation time. Increasing the initial L-value above 2 achieves at best an insignificant objective improvement of only 0.01% and this is at great computational expense. We therefore use the refined algorithm with an initial value of $L = 2$ when testing the algorithm in the next section.



(a) Tuning for small instances

(b) Tuning for large instances

Figure 7.3: Tuning the algorithm on data sets of two different sizes

7.7 Computational Results

In order to explore the benefits of integrating bunker planning in the routing and scheduling phase, we compare the devised solution method with the standard sequential approach where routes and schedules are planned with no consideration to actual bunkering. When planning routes and schedules in this standard approach, bunker consumption is accounted at the average of all bunker prices valid at the time of planning and no actual bunker stops are planned, meaning that no time is scheduled for bunkering and no bunker port costs are incurred. Each optimal schedule from this process now assigns a given cargo set to each ship and a bunker plan must be created that respects this cargo assignment to ships. We find the optimal bunker plan by fixing cargoes to ships according to this cargo assignment, and then run the bunker algorithm with $L = 17$.

When testing the devised solution method, we consider the same fixed fleet of 7 ships as when tuning in Section 7.6 and also use the same 38 ports for all test instances. We have used the instance generator described in Section 7.5 to generate 25 cargo instances. With a pickup time horizon of 30 days we have five sets with 30 cargoes, five sets with 40 cargoes and five sets with 50 cargoes. For the 60 days horizon we have five sets with 50 cargoes and five sets with 60 cargoes. We have constructed two versions of each of these 25 cargo instances: one where all cargoes are spot cargoes and one where 15% of the cargoes are randomly chosen to be contract cargoes. The sets without contract cargoes allow us to explore how much the integrated planning approach can at best increase profits while the other sets can help estimate the expected decrease in profit gain when incorporating contract cargoes. We also generated problem instances with a planning horizon of 90 days but when testing on these instances we found that the sequential approach in 2 out of 3 cases produced a solution that was infeasible with respect to bunkering. By construction, the integrated approach found feasible solutions for all instances. Therefore, we could not compare the two methods on these cases and do not report them here. It should be noted that bunkering becomes more relevant the longer the planning horizon we consider as ships must travel more, but at the same time the assumption of valid price forecasts becomes more unrealistic.

The bunker price generator has been used to generate six bunker instances: Three price instances for the 30 days planning horizon with 14 weekly bunker options per port, i.e. a total of 532 bunker options, and three price sets for the 60 days horizon with 19 bunker options per port, i.e. 722 bunker options. Across all six price instances and across all time periods the prices range from \$494 per tonne to \$705 with an average price of \$608. The maximum recorded price spread within a time period is \$183 while the average and minimal spread is, respectively, \$148 and \$119. Within the individual regions the maximum and minimum recorded spread is \$120 and \$21, respectively. On average the spread for a given time period constitutes 22% of the maximum price for the time period and 24% of the average price of the time period.

These cargo and price instances can be combined to a total of 2×75 problem instances. Table 7.3 gives an overview of these problem instances of varying size and complexity. We use the same notation as in Section 7.6 and denote a problem instance with e.g. 40 cargoes over a pickup horizon of 30 days with 14 bunker options per port by C40/PH30/B14.

Table 7.3: Problem instance overview

Ships	7	7	7	7	7
Ports	38	38	38	38	38
Cargoes	30	40	50	50	60
Pickup Time Horizon (days)	30	30	30	60	60
Bunker options per port	14	14	14	19	19
Bunker options in total	532	532	532	722	722
Number of instances	2×15	2×15	2×15	2×15	2×15

On each of these 150 problem instances we have run both the standard sequential approach described above and the integrated approach defined by the refined bunker algorithm described in Section 7.4 and Section 7.6 with $L = 2$ afterwards increased to $L = 17$. All computational experiments were performed on a PC with 4.0 GB RAM and an Intel(R) Core(TM)2 Duo CPU P8600, 2.4 GHz processor under a 64 bit Windows 7. Both algorithms were entirely developed in C++ using Cplex 12.4 with default settings to solve the master problem.

Table 7.4 summarizes some key values for the refined bunker algorithm on the problem instances containing only spot cargoes while Table 7.5 does the same for the instances with a mix of spot and contract cargoes. Each line corresponds to the average key values over the 15 problem instances of the corresponding problem type given by the entry in the left most column. We do not report the objective function value but return to that when comparing with the sequential approach. The key values reported are, respectively, the percentage gap from the forced integer solution to the LP solution (Gap), CPU seconds for the whole algorithm ($\text{CPU}_{\text{Total}}$), CPU seconds for reoptimizing bunker with $L = 17$ for fixed routes (CPU_{17}), CPU seconds for solving all subproblems in the column generation phase with $L = 2$ (CPU_{Sub}), the number of columns generated (Cols.) and the number of calls to the subproblems (Subs) (i.e. number of iterations) in the column generation procedure with $L = 2$, the percentage of all bunker stops that corresponded to filling up to tank capacity (Filled), the number of cargoes carried in the final solution (Cargoes) and finally, some statistics on price sensitivity (PS(Av,Max)). As each cargo instance is run with three different price instances we can get an idea of how sensitive the method is to changes in prices. For this we consider the differences in carried cargoes between two solutions derived from the same cargo instance but from different price instances. If one solution carries x more cargoes than the other solution, we define the *cargo difference* to be equal to x. Two solutions carrying the same number of cargoes do not necessarily carry the *same* cargoes and we increase the cargo difference count by one for each difference in carried cargoes when comparing the two solutions. As a small example, imagine that one solution carries cargoes 1, 2 and 3 while another solution carries cargoes 1, 3, 4 and 5. Such a solution would correspond to one extra cargo *and* one different cargo and we would therefore define the cargo difference between these two solution to be equal to two. In the price sensitivity column (denoted PS) we report the average and the maximum cargo difference when comparing solutions derived from the same cargo instance.

Table 7.4: Key values for the refined bunker algorithm on spot cargo sets

	Gap	$\text{CPU}_{\text{Total}}$	CPU_{L17}	CPU_{Sub}	Cols	Subs	Filled	Cargoes	PS(Av,Max)
C30/PH30/B14	0.00	17.1	6.8	9.9	205	6	70.0	16.5	(1.2 , 2)
C40/PH30/B14	-	27.3	10.5	16.3	248	6	75.6	18.3	(0.9 , 2)
C50/PH30/B14	0.22	44.8	14.0	30.1	287	7	71.8	20.9	(2.2 , 5)
C50/PH60/B19	0.26	123.9	33.9	88.8	453	11	65.3	29.0	(3.8 , 8)
C60/PH60/B19	0.07	167.4	37.9	128.1	513	11	64.5	30.0	(3.9 , 8)

Table 7.5: Key values for the refined bunker algorithm on mixed cargo sets

	Gap	CPU_{Total}	CPU_{L17}	CPU_{Sub}	Cols	Subs	Filled	Cargoes	PS(Av,Max)
C30/PH30/B14	0.02	21.8	8.4	13.0	250	8	72.7	16.3	(0.5 , 3)
C40/PH30/B14	-	32.4	11.7	20.2	356	8	74.1	18.3	(0.5 , 2)
C50/PH30/B14	0.07	53.8	14.2	38.8	430	9	73.9	20.5	(1.1 , 4)
C50/PH60/B19	0.38	152.0	36.5	114.2	728	15	66.6	29.6	(1.3 , 4)
C60/PH60/B19	0.21	222.7	44.1	177.1	733	15	65.0	30.8	(1.6 , 6)

Out of the 75 spot cargo instances, we obtained fractional solutions from only 19 instances while there were 28 fractional solutions for the 75 mixed cargo instances. However, from Table 7.4 and 7.5 we note that the integrality gap is relatively small. Aside from justifying our non optimal integer approach this also suggests that the ships are not competing for the cargoes. This is probably because the fleet operates in a very large part of the world, and, hence, ports are spread over vast distances. For a given cargo, chances are that only one available ship is close enough for it to be profitable to carry the cargo. Vice versa, for a given ship, there are only a few cargoes that are both reachable with respect to time but also profitable. The addition of contract cargoes introduces a form of dependency between the ships because they must share these cargoes somehow. This dependency explains the increase in fractional solutions and also the added problem complexity reflected by the extra running time of the algorithm on instances with mixed cargoes.

From Table 7.4 and 7.5 we also see that the majority of bunker stops correspond to filling up to tank capacity and that this number seems to be unaffected by the addition of contract cargoes. Finally, from the price sensitivity column we see that the optimal solution is indeed affected by changes in prices though the effect is, as could be expected, smaller when contract cargoes are introduced. For the larger instances, two solutions from the same spot cargo instance can differ by as much as 8 cargoes making an accurate price forecast very important.

Before considering the sequential approach we first present some network statistics in Table 7.6 for the bunker optimization with $L = 17$. The numbers are unaffected by the introduction of contract cargoes so we only present one table. In the first part of the table we report statistics on the actual network sizes: The number of nodes in the average subproblem network (Nodes), the number of these that were bunker nodes (bNodes), the arcs in the average network (Arcs) and, finally, the number of these that where bunker arcs (bArcs). In the second part of the table we report the potential network sizes of, respectively, the aggregated (agNodes and agArcs) and the extended (extNodes and extArcs) cargo-bunker networks as stated in Section 7.4.4.

Table 7.6: Network statistics for the refined bunker algorithm

	Actual Network Size				Potential Network Size			
	Nodes	bNodes	Arcs	bArcs	agNodes	agArcs	extNodes	extArcs
C30/PH30/B14	19,362	19,303	39,423	39,252	64,966	1,055,384	989,582	1,980,000
C40/PH30/B14	27,612	27,534	56,974	56,688	86,266	1,832,824	1,745,042	3,491,600
C50/PH30/B14	35,833	35,736	77,165	76,735	107,566	2,823,264	2,713,302	5,429,000
C50/PH60/B14	49,899	49,802	150,177	149,482	145,946	3,830,644	3,682,302	7,367,000
C60/PH60/B14	57,588	57,474	209,821	208,849	174,846	5,463,484	5,285,162	10,573,800

We first note from Table 7.6 that bunker nodes constitute over 99.7% of the total nodes in the networks while the corresponding number for the arcs is 99.5%. Next, we see that preprocessing has allowed a network node reduction of 66-70% compared to the potential network size stated in

Section 7.4.4. Similarly, preprocessing has removed 96-97% of the arcs. When comparing with the extended cargo-bunker network illustrated in Figure 7.2(a), we see that the potential node count of the aggregated networks that we use, is 93-97% lower than that of the corresponding extended networks while the arc count is 47-48% lower.

The key values for the standard sequential approach on the 2×75 instances are reported in Table 7.7 and 7.8. They are almost the same as for the refined bunker algorithm but the CPU time for solving subproblems now corresponds to the column generation phase of finding routes and schedules without optimizing bunker simultaneously. Likewise, the number of generated columns and the number of calls to the subproblems are derived from the pure routing and scheduling phase. Finally, we do not report any values on price sensitivity as the bunker prices are not considered while constructing the routes and schedules when using the sequential approach.

Table 7.7: Key values for sequential algorithm on spot cargo sets

	Gap	CPU_{Total}	CPU_{L17}	CPU_{Sub}	Cols	Subs	Filled	Cargoes
C30/PH30/B14	-	7.9	7.8	0.0	88	6	69.7	16.8
C40/PH30/B14	-	10.4	10.3	0.0	94	5	76.0	18.4
C50/PH30/B14	0.25	13.7	13.5	0.0	121	6	69.5	20.8
C50/PH60/B19	0.22	30.3	30.1	0.0	196	10	65.6	28.8
C60/PH60/B19	0.12	44.4	44.1	0.0	219	9	65.2	30.4

Table 7.8: Key values for sequential algorithm on mixed cargo sets

	Gap	CPU_{Total}	CPU_{L17}	CPU_{Sub}	Cols	Subs	Filled	Cargoes
C30/PH30/B14	-	8.3	8.2	0.0	95	7	72.0	16.2
C40/PH30/B14	-	11.3	11.2	0.0	144	7	73.7	18.2
C50/PH30/B14	0.25	13.8	13.7	0.0	180	8	73.6	20.4
C50/PH60/B19	0.36	35.8	35.6	0.0	304	14	65.3	29.6
C60/PH60/B19	0.28	45.0	44.7	0.1	319	14	64.2	31.0

On 2 out of the 25 mixed cargo instances the sequential approach produced a solution that was infeasible with respect to bunkering. The infeasibility is of course independent of bunker prices and therefore 6 bunker infeasible solutions were produced.

For the sequential approach, 24 out of the 75 spot cargo instances resulted in a fractional solution while for the mixed cargo instances the same number was 18. However, as can be seen in Table 7.7 and 7.8 the integrality gap is again relatively small. Furthermore, we see that bunker optimization, i.e. rerunning with bunkering included and $L = 17$, is accountable for almost all the CPU time and that the introduction of contract cargoes slightly increases the running time.

Finally, in Table 7.9 we compare the two approaches to see the effect of integrating bunker. Each entry in the table corresponds to the average over the 15 problem instances generated for the given problem category. For the columns marked with an * we only report the average over 12 instances as the remaining 3 instances resulted in a bunker infeasible solution when using the sequential approach. The objective function value for the sequential approach serves as a base at which we compare the objective value from the integrated approach. Therefore, the objective values for the sequential approach (Obj) are not reported, and the objective values for the integrated approach (Obj%) are given as the percentage increase from the corresponding sequential objective function values. We do not report the actual objective function values since these are to some extent artificial due to the inclusion of bonus bunker. Both algorithms, however, include this and therefore we can still compare their objective function values. For both algorithms we report the CPU seconds for running the entire algorithm (CPU), the number of cargoes carried in the final solution (Cargoes),

and the number of bunker stops in the final solution (Bunker). In the lower part of the table we report the average (Av. Cargo Diff.) and maximum (Max Cargo Diff.) cargo difference when comparing the solutions found by the integrated approach with those of the sequential approach.

Table 7.9: Comparing the two planning approaches

	Spot cargo sets					Mixed cargo sets				
	PH30/B14			PH60/B19		PH30/B14			PH60/B19	
	C30	C40	C50	C50	C60	C30*	C40	C50	C50	C60*
Sequential										
Obj	-	-	-	-	-	-	-	-	-	-
CPU	7.9	10.4	13.7	30.3	44.4	8.4	11.3	13.8	35.8	49.8
Cargoes	16.8	18.4	20.8	28.8	30.4	16.3	18.2	20.4	29.6	31.8
Bunker	13.2	14.7	14.7	21.3	21.5	12.5	14.2	15.1	20.9	21.2
Integrated										
Obj%	0.4	0.3	0.7	0.5	0.6	0.1	0.1	0.2	0.5	0.4
CPU	17.1	27.3	44.8	123.9	167.4	22.2	32.4	53.8	152.0	240.2
Cargoes	16.5	18.3	20.9	29.0	30.0	16.3	18.3	20.5	29.6	31.5
Bunker	13.3	14.7	16.1	21.3	21.6	12.7	14.1	14.8	21.3	21.0
Av. Cargo Diff.	0.8	0.9	2.7	3.7	3.7	0.3	0.3	1.1	2.0	1.9
Max Cargo Diff.	2	3	5	6	8	3	2	4	3	7

When comparing the two planning approaches, we see that the percentage increase in profit is relatively small and that the increase is smaller when contract cargoes are introduced. It is however, important to remember that fixed costs have not been subtracted; hence, we are actually comparing marginal contributions rather than profits. The actual profits will therefore be much lower and any difference in profits will correspond to a larger percentage. We, however, do not have data on fixed costs and so, we use the marginal contributions as above. We also note that the profits obtained from the sequential approach are expected to be an optimistic estimate of the standard sequential approach where current practice is to use manual planning in both phases. Furthermore, we note that we are dealing with an industry where numbers are huge. This means that even small percentage increases can lead to huge increases in profit. Finally, we note that including more ports can help increase the bunker effect as distances between ports will become smaller. As already mentioned, with our setup the distances between ports, and in turn between cargoes, are often so large, that for a given ship, only very few cargoes are actually eligible for transportation.

From Table 7.9 we also note that the integrated approach does not in general produce solutions that carry more cargoes than the sequential approach. The method is not designed to increase fleet utilization in the sense of carrying extra cargoes. Rather it is designed to increase fleet utilization by carrying the right cargoes and we see that the cargo difference can be as high as 8 cargoes for the spot cargo instances and 7 for the mixed cargo instances. Aside from the reported cargo differences, we very often found that cargoes carried in both the sequential solution and the integrated solution where carried by different ships in the two solutions.

Table 7.9 also shows that the two solution approaches generally produce solutions that have the same number of bunker stops. However, we note that the solutions produced by the sequential approach generally rely on more pure bunker stops (i.e. ports where the ships only bunker) than the integrated approach. This is to be expected as the sequential approach does not factor in bunker prices when selecting the cargoes to carry.

Overall, we note that a small profit increase can be obtained at little computation time by integrating bunkering in the routing and scheduling planning phase. It should also be noted that such an integration will prevent the construction of bunker infeasible schedules as we saw for many of the larger instances.

7.8 Concluding Remarks

In this paper we have considered the tramp ship routing and scheduling problem with simultaneous bunker optimization for full shiploads. We have presented a mixed integer programming formulation that extends the standard tramp formulation by accounting for bunkering time, variations in bunker prices and bunker port costs. We have also extended standard formulations by using load dependent cost, speed and bunker consumption. We have developed a solution method that utilizes column generation with a dynamic programming algorithm to generate columns. The devised method is heuristic and this is mainly due to the discretization of the continuous bunker purchase variables. Results from tuning in Section 7.6 show that there is very little gain from refining this discretization and therefore it seems that solving the continuous version of the problem can at best yield very small improvements in the solutions. Likewise, our computational results show that for most instances we obtain integral solutions and that the integrality gap is small for the fractional solutions. Therefore, embedding the column generation scheme in a Branch & Bound framework is not expected to change results much. Generally, the method is very flexible and can be extended to incorporate various operator specific characteristics such as e.g. multiple product types, tank cleaning and restrictions on product successions, by simply changing the subproblem networks and the corresponding labels and resource extension functions.

We have compared the method with a standard sequential approach where routes and schedules are planned without considerations for bunkering. Computational results on 150 generated test instances show that the integrated approach can increase profits slightly. They also show that the decision of which cargoes to carry and on which ships is affected by the bunker integration and by changes in the bunker prices. Consequently, we recommend combining the decisions on fleet scheduling and bunker optimization rather than separating the two planning problems as is current practice.

We also want to mention that the work presented here assumes only one type of bunker even though several exist in practice. It would therefore be very interesting to extent this work to consider multiple types of bunker. Finally, we have solved the problem using a forward curve for the bunker price at each bunker port. If in fact several price scenarios exist, it would be interesting to apply stochastic programming to cope with this price uncertainty.

Acknowledgements

The research presented in this paper has been partly funded by The Danish Maritime Fund and we gratefully acknowledge their financial support. The authors also wish to thank the experienced staff at Maersk Tankers A/S for fruitful discussions and for helping us gain insight into the tramp shipping industry. In particular we would like to thank Jakob Tørring for helpful advice, constructive critique and general support during the project.

Bibliography

K. Abdelghany, A. Abdelghany, and S. Raina. A model for the airlines' fuel management strategies. *Journal of Air Transport Management*, 11(4):199–206, 2005.

L.H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3:53–68, 1969.

O. Besbes and S. Savin. Going bunkers: The joint route selection and refueling problem. *Manufacturing and Service Operations Management*, 11(4):694–711, 2009.

G. Brønmo, M. Christiansen, and B. Nygreen. Ship routing and scheduling with flexible cargo sizes. *Journal of the Operational Research Society*, 58(9):1167–1177, 2007.

G. Brønmo, B. Nygreen, and J. Lysgaard. Column generation approaches to ship scheduling with flexible cargo sizes. *European Journal of Operational Research*, 200(1):139–150, 2010.

M. Christiansen and B. Nygreen. Robust inventory ship routing by column generation. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, chapter 7, pages 197–224. Springer, New York, 2005.

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

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

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

D.W. Darnell and C. Loflin. National airlines fuel management and allocation model. *Interfaces*, 7(2):1–16, 1977.

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

G. Desaulniers, J. Desrosiers, A. Erdmann, M.M. Solomon, and F. Soumis. Vrp with pickup and delivery. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 9, pages 225–242. Society for Industrial and Applied Mathematics, 2002.

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

J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Routing. Handbooks in Operations Research and Management Science, vol. 8*, chapter 2, pages 35–139. North-Holland, Amsterdam, 1995.

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

F. Hennig, B. Nygreen, and M.E. Lübbecke. Nested column generation applied to the crude oil tanker routing and scheduling problem with split pickup and split delivery. *Naval Research Logistics*, 59(3-4):298–310, 2012.

I. Ioachim, S. Gélinas, F. Soumis, and J. Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204, 1998.

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

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

K. Kang, W.-C. Zhang, L.-Y. Guo, and T. Ma. Research on ship routing and deployment mode for a bulk. *International Conference on Management Science and Engineering - Annual Conference Proceedings*, pages 1832–1837, 2012.

H.-J. Kim, Y.-T. Chang, K.-T. Kim, and H.-J. Kim. An epsilon-optimal algorithm considering greenhouse gas emissions for the management of a ship's bunker fuel. *Transportation Research Part D: Transport and Environment*, 17(2):97–103, 2012.

K. Kobayashi and M. Kubo. Optimization of oil tanker schedules by decomposition, column generation, and time-space network techniques. *Japan Journal of Industrial and Applied Mathematics*, 27(1):161–173, 2010.

S.-H. Lin. Finding optimal refueling policies in transportation networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5034 LNCS:280–291, 2008.

S.H. Lin, N. Gertsch, and J.R. Russell. A linear-time algorithm for finding optimal vehicle refueling policies. *Operations Research Letters*, 35(3):290–296, 2007.

T. E. Notteboom and B. Vernimmen. The effect of high fuel costs on liner service configuration in container shipping. *Journal of Transport Geography*, 17(5):325–337, 2009.

H.C. Oh and I.A. Karimi. Operation planning of multiparcel tankers under fuel price uncertainty. *Industrial and Engineering Chemistry Research*, 49(13):6104–6114, 2010.

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

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

D. Ronen. Marine inventory routing: Shipments planning. *Journal of the Operational Research Society*, 53(1):108–114, 2002.

J. Schönberger, H. Kopfer, and D.C. Mattfeld. A combined approach to solve the pickup and delivery selection problem. In U. Leopold-Wildburger, F. Rendl, and G. Wäscher, editors, *Operations Research Proceedings 2002*, pages 150–155. Springer, 2003. Selected Papers of the International Conference on Operations Research (SOR 2002), Klagenfurt, September 2-5, 2002.

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

J.S. Stroup and R.D. Wollmer. Fuel management model for the airline industry. *Operations Research*, 40(2):229–237, 1992.

Y. Suzuki. A generic model of motor-carrier fuel optimization. *Naval Research Logistics*, 55(8): 737–746, 2008.

Y. Suzuki and J. Dai. Reducing the fuel cost of motor carriers by using optimal routing and refueling policies. *Transportation Journal*, 51(2):145–163, 2012.

L. Tang, X.-L. Xie, and C.-W. Wang. Model of tramp ship scheduling with variable speed based on set partition approach. *Shanghai Jiaotong Daxue Xuebao/Journal of Shanghai Jiaotong University*, 47(6):909–915, 2013.

UNCTAD. Review of maritime transport 2011. `http://unctad.org/en/Pages/PublicationArchive.aspx?publicationid=1734`, November 2011.

S. Wang, Q. Meng, and Z. Liu. Bunker consumption optimization methods in shipping: A critical review and extensions. *Transportation Research Part E: Logistics and Transportation Review*, 53(1):49–62, 2013.

Z. Yao, S.H. Ng, and L.H. Lee. A study on bunker fuel management for the shipping liner services. *Computers and Operations Research*, 39(5):1160–1172, 2012.

P.P. Zouein, W.R. Abillama, and E. Tohme. A multiple period capacitated inventory model for airline fuel management: a case study. *Journal of the Operational Research Society*, 53:379–386, 2002.

Chapter 8

A Heuristic and Hybrid Method for the Tank Allocation Problem in Maritime Bulk Shipping

Charlotte Vilhelmsen Jesper Larsen Richard M. Lusby

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

Abstract In bulk shipping, ships often have multiple tanks and carry multiple inhomogeneous products at a time. When operating such ships it is therefore a major challenge to decide how to best allocate cargoes to available tanks while taking into account tank capacity, safety restrictions, ship stability and strength as well as other operational constraints. The problem of finding a feasible solution to this tank allocation problem has been shown to be *NP-Complete*. We approach the problem on a tactical level where requirements for computation time are strict while solution quality is less important than simply finding a feasible solution. We have developed a heuristic that can efficiently find feasible cargo allocations. Computational results show that it can solve 99% of the considered instances within 0.4 seconds and all of them if allowed longer time. We have also modified an optimality based method from the literature. The heuristic is much faster than this modified method on the vast majority of considered instances. However, the heuristic struggles on two instances which are relatively quickly solved by the modified optimality based method. These two methods therefore complement each other nicely and so, we have created a hybrid method that first runs the heuristic and if the heuristic fails to solve the problem, then runs the modified optimality based method on the parts of the problem that the heuristic did not solve. This hybrid method cuts between 90% and 94% of the average running times compared to the other methods and consistently solves more instances than the other methods within any given time limit. In fact, this hybrid method is fast enough to be used in a tactical setting.

8.1 Introduction

Every year 8.7 billion tons of goods or equivalently 80% of world trade by volume is carried by ships (UNCTAD, 2012). This translates into well over a tonne of cargo for every single individual on the planet, every single year, and the global economy therefore depends on the international shipping industry's efficiency and competitive freight rates. Hence, research to increase efficiency within maritime transportation is important, and, taking the mere size of this huge industry into consideration, even small improvements can have great impact.

In this paper we consider maritime bulk shipping, both wet and dry. Many bulk ships have multiple tanks and can thereby carry multiple inhomogeneous products at a time. Two examples of such ships are oil product tankers and chemical tankers. A major challenge when operating such ships is how to best allocate cargoes to available tanks while taking tank capacity, safety restrictions for onboard cargoes, ship stability and strength, as well as other operational constraints into account.

The complexity of the allocation problem varies with the number of tanks and the number and type of different products transported simultaneously. A chemical tanker can for instance have as many as 50 different tanks and hazmat (hazardous materials) regulations play a major role when allocating the products to the different tanks. E.g. products in neighboring tanks must be non-reactive and incompatible products must not succeed each other in a tank unless it is cleaned (this can be costly). The regulations on product succession mean that we need to keep track of previous tank allocations and that decisions at any voyage leg affect decisions at future voyage legs complicating the problem even further. Often it is not allowed to move a cargo once it has been allocated to tanks and then this interdependency between voyage legs becomes even stronger. Taking stability, safety restrictions and other operational constraints into consideration it can therefore be extremely difficult, if not impossible, to find a feasible allocation for a given set of cargoes. In fact, Hvattum et al. (2009) show that the problem of finding a feasible solution is *NP-Complete*.

The *Tank Allocation Problem* (TAP) as described above is an operational planning problem, which is normally solved for a given route, i.e. *after* the fleet routes and schedules have been constructed at the tactical planning level. However, separating these two planning problems means that we can potentially create routes and schedules for which no feasible tank allocation exists. Therefore, for bulk ship fleets the tank allocation aspect should be included in the routing and scheduling phase of planning. Tank cleaning costs and other costs related to tank allocations are, however, insignificant compared to the profits from carrying cargoes. Hence, at the tactical planning level where routes and schedules are determined, we can simplify the tank allocation aspect by ignoring the allocations costs and simply focus on finding a feasible allocation. Note that this also means that we can refrain from keeping track of past cargo allocations since we can just assume that all tanks are cleaned whenever needed.

Within shipping and many other areas, routing and scheduling problems are often solved in a way that requires assessment of numerous routes, as for instance in column generation and local search based methods. For each considered route, the TAP must be solved to assess route feasibility with respect to stowage. The solution time for the entire procedure will therefore only be acceptable if the TAP can be solved efficiently. Furthermore, uncertainty plays a big part in maritime optimization where planners face a constantly changing environment with large daily variations in demand and many unforeseen events and so, it is often necessary to re-plan routes and schedules continuously to accommodate new cargoes and changes to existing plans. In effect, this means that the TAP must be solved repeatedly and that the requirements for computation time are strict.

Bulk ship operators of large and even medium size can easily have fleets with more than 25 ships. Since these ships can carry multiple cargoes onboard simultaneously, the combinatorial puzzle associated with routing the ships is much larger than for ships sailing full shiploads, i.e. having just one cargo onboard at a time. Thereby, there can be easily be more than 200 routes to evaluate for each ship and so, it is necessary to assess feasibility with respect to stowage for at least $25 \cdot 200 = 5000$ routes. If we allow a run time of up to just 0.25 seconds, assessing feasibility for these routes can alone take $0.25 \cdot 5000 = 1250$ seconds, i.e. 21 minutes. Hence, if we want to develop a method that is applicable to bulk ship operators of all sizes, the requirements for computation are quite strict.

Hvattum et al. (2009) find that constraint programming fails to solve the TAP mainly because of the stability constraints. Instead they provide a a mixed integer formulation for the problem and use a commercial solver to solve it. However, their running times are much too long to allow this method to be used in a tactical setting and they specifically advocate for development of a heuristic method for determining feasibility of the TAP. In this paper, we update and modify their method and this yields a significant improvement on running times. However, even with this improvement running times are still a bit too long.

Neo et al. (2006) solves the integrated problem of routing a fleet of multi-compartment tankers with the tank allocation aspect included. They present an integer programming model for this problem and use a commercial solver to solve it. However, even for a small instance with just a single ship and only 10 potential cargoes, they report running times above 18,000 seconds. The aim of our work is therefore to develop a heuristic method that can facilitate the incorporation of the tank allocation aspect into the routing and scheduling planning phase by efficiently finding feasible cargo allocations for given ship routes.

In this paper we explore the TAP from a tactical perspective where the main objective is to quickly assess feasibility of a given ship route rather than finding an optimal tank allocation. Our main contribution is a heuristic solution method for efficiently finding feasible cargo allocations. Computational results show that it can solve 99% of the considered instances within 0.4 seconds and all of them if allowed longer running time. The heuristic does however struggle on two instances causing an overall longer average running time than found with an optimality based method from the literature. However, when running time is below 10 seconds, our heuristic clearly outperforms the optimality based method by consistently solving more instances. Two further contributions of this work are therefore the modification of this optimality based method and a hybrid method that combines the developed heuristic with this modified method. Computational results show that on the considered instances this hybrid method cuts between 90% and 94% of average running times compared to the other methods and consistently solves more instances than the other methods within any given time limit. The average running time for the hybrid method is just 0.027 seconds. Hence, we have developed a solution method that is efficient enough to allow the inclusion of the tank allocation aspect into the routing and scheduling phase.

The remainder of the paper is organized as follows. Section 8.2 provides a problem description as well as a mathematical model for the problem and gives an overview of the existing literature related to the TAP. The devised heuristic is described in Section 8.3, while section 8.4 describes the data used to evaluate its performance. In Section 8.5 we tune the heuristic and in Section 8.6 we compare it to an optimality based method and a modified version of this method. Furthermore, we explore the effect of combining our heuristic with the modified version of the optimality based method into a hybrid solution method. Finally, concluding remarks and suggestions for future work are discussed in Section 8.7.

8.2 Problem description

In this paper we consider the TAP as it is described and modelled in Hvattum et al. (2009). However, we approach the problem on a tactical level, where the focus is on feasibility rather than optimality.

For each instance, a fixed route is given for a specific ship with a number of tanks (or compartments). Besides the capacity of a tank, also the material and the coating can have an impact on how it can be used. A ship route is a collection of voyage legs and each voyage leg has a departure port where one cargo is picked up and a destination port where one cargo is discharged, though not necessarily the same cargo as the one that was picked up. The destination port of one voyage leg is the departure port of the next voyage leg. Note that both the heuristic and hybrid method described in Section 8.3 also work in situations where multiple cargoes can be picked up and discharged in each port.

In addition to the fixed ship route, a set of cargoes that must be carried is also given. For each cargo, the volume and the density of it is given, and in addition also the ports where it is picked up and discharged, respectively. If it is allowed to move cargoes between tanks after they have been loaded onboard the ship, the full route problem can be reduced to multiple instances of the Single Instance Tank Allocation Problem (SITAP). In the SITAP the allocation problem is solved for a set of cargoes on a single voyage leg. However, normally it is not allowed to move cargoes once they have been allocated and certainly, it will always be undesirable to so, due to both time consumption and possibly additional tank cleaning costs. Therefore, here we assume that it is not allowed to move cargoes once they have been allocated to the ship. When we define a problem instance, some cargoes may already be onboard the ship at the beginning of the planning period and thereby occupy some of the tanks.

The tactical version of the TAP can now formally be defined: Given a specific ship, a fixed ship route, and a set of cargoes, find a feasible allocation of cargoes to tanks on the ship. A feasible tank allocation is constrained by a number of operational factors that can be divided into three main groups:

- **Tank usage**: Each cargo must be allocated to one or several tanks and the cargo volume cannot exceed the total capacity of the allocated tank(s). A cargo can only go into a tank if it is compatible with the coating. It is not allowed to mix different products in the same tanks. Even if multiple cargoes consists of the same material, they cannot be mixed but must remain in separate tanks. For liquid products, a minimum volume must be allocated to used tanks in order to avoid excessive sloshing at sea.

- **Ship stability**: Ship stability and strength in all dimensions must be maintained throughout the ship's route.

- **Hazmat rules**: Due to hazmat rules, certain materials cannot be allocated to neighboring tanks and certain materials cannot follow each other in a tank unless it is cleaned.

For completeness we here present the model derived and thoroughly described in Hvattum et al. (2009), though with slightly modified notation. We have the following sets:

C	Set of cargoes
T	Set of tanks
C_c^C	Set of cargoes in conflict with cargo c
C_t^T	Set of cargoes compatible with tank t
T_c^C	Set of tanks compatible with cargo c
T_{ckt}	Set of tanks which cannot be used for cargo k if cargo c is in tank t
N_c	Set of cargoes on board the ship immediately after adding cargo c
P_c	Set of all cargoes that have been present on the ship before adding cargo c
Q	Set of subsets; each set corresponds to the cargoes present on the ship at a time when the stability and strength restrictions apply.
S	Set of stability and strength dimensions, e.g. trim and roll

For each cargo c, the cargo volume v_c and the cargo density d_c is given. The capacity of tank t is denoted κ_t while the minimum volume in the tank when used is denoted b_t. Furthermore, m_t^s denotes the moment arm for tank t with respect to stability or strength dimension $s \in S$ while m^{s+} and m^{s-} denote, respectively, the upper and lower limit on total moment for stability or strength dimension $s \in S$. Finally, h_{ckt} denotes the minimum number of compatible cargoes that must be allocated to tank t before cargo c can be allocated to tank t if a cargo $k \in C_c^C$ that is incompatible with cargo c has previously been allocated to tank t.

We have three sets of decision variables. The first, x_{ct}, is a binary decision variable equal to 1 if cargo c is allocated to tank t. The second, y_{ct}, is a continuous variable indicating the volume of cargo c in tank t. Finally, z_{ct} is a binary decision variable which is equal to 1 if tank t is cleaned just before adding cargo c.

A feasible solution can now be defined by the following constraints:

$$y_{ct} \leq \kappa_t x_{ct}, \qquad \forall c \in C,\, t \in T_c^C, \quad (8.1)$$

$$b_t x_{ct} \leq y_{ct}, \qquad \forall c \in C,\, t \in T_c^C, \quad (8.2)$$

$$\sum_{t \in T_c^C} y_{ct} = v_c, \qquad \forall c \in C, \quad (8.3)$$

$$\sum_{k \in C_t^T \cap N_c} x_{kt} \leq 1, \qquad \forall c \in C,\, t \in T, \quad (8.4)$$

$$\sum_{k \in C_c^C \cap N_c} \sum_{u \in T_{ckt}} x_{ku} \leq M_{ct}(1 - x_{ct}), \qquad \forall c \in C,\, t \in T_c^C, \quad (8.5)$$

$$m^{s-} \leq \sum_{c \in R} \sum_{t \in T_c^C} m_t^s d_c y_{ct} \leq m^{s+}, \qquad \forall R \in Q,\, s \in S, \quad (8.6)$$

$$h_{ckt}(x_{ct} - z_{ct}) - \sum_{j \in R}(x_{jt} + h_{ckt} z_{jt}) \leq h_{ckt}(1 - x_{kt}), \qquad \forall c \in C, t \in T_c^C, k \in P_c \cap C_c^C, R = P_c \setminus P_k \setminus \{k\}, \quad (8.7)$$

$$x_{ct} \in \{0, 1\}, \qquad \forall c \in C,\, t \in T_c^C, \quad (8.8)$$

$$y_{ct} \geq 0, \qquad \forall c \in C,\, t \in T_c^C, \quad (8.9)$$

$$z_{ct} \in \{0, 1\}, \qquad \forall c \in C,\, t \in T_c^C. \quad (8.10)$$

Due to our focus on feasibility, we have not presented an objective function. However, a possible objective function could be the minimisation of tank cleaning or the maximisation of vacant tank capacity in order to preserve flexibility to accommodate future cargoes or cargo changes.

Constraints (8.1) ensure that the capacity of each tank is not exceeded, and that only cargoes allocated to a given tank can be put into that tank. Constraints (8.2) ensure that an occupied tank is allotted a minimum content to avoid sloshing. In addition, constraints (8.3) and (8.4) make sure that the entire cargo is placed on the ship and prohibit more than one cargo per tank at a time. Constraints (8.5) make sure that the hazmat rules regarding what chemicals can be put next to each other is maintained. Here, M_{ct} is a large constant which can be set as $\max_k\{T_{ckt}\}, c \in C, t \in T_c^C$. Constraints (8.6) ensure that the ship remains sufficiently balanced with regards to roll, trim and strength on all legs of the route. Note that these constraints are simplified versions of the actual non-linear expressions. Hvattum et al. (2009) use this same simplification and argue that the loss from it is small. Constraints (8.7) restrict a cargo c from going into a tank t if the previous cargo k is incompatible with c. Exceptions to this can be allowed if the tank is cleaned or h_{lkt} other cargoes occupy the tank in between k and c. Note that by ignoring the tank cleaning costs, we can actually ignore these constraints and simply assume that all tanks are cleaned. However, we have included them here for the sake of completeness. Finally, constraints (8.8), (8.9), and (8.10) formally define the variables of the problem. In (Hvattum et al., 2009) a complexity analysis concludes that the TAP is *NP-Complete* by showing that it is a generalisation of the segregated storage problem that has been shown to be *NP-Complete*.

A more simple variant of the problem was already introduced in (Vouros et al., 1996). Here an expert system is presented for the task of allocating cargoes to a ship for one port. So, given the current configuration how does one load a cargo on board the ship in the best possible way adhering to ship stability constraints and hazmat rules. The paper only describes the approach, neither testing nor implementation details are presented.

As we have already mentioned a number of times, Hvattum et al. (2009) first formally describe the variant of the TAP that we base our work upon. They solve their mixed integer formulation with a commercial solver but report running times that are far beyond what would be acceptable in a tactical setting. They also try to solve the problem using constraint programming. However, their constraint solver failed to find a feasible solution to even one of the instances. Therefore, they note in their conclusion that a heuristic method for this problem is of interest. The first description of the routing problem for multi-compartment tankers is given in Jetlund and Karimi (2004). This paper presents the problem of optimising the route of a single multi-compartment tanker as well as

a fleet of ships. Although the paper presents the routing problem for a multi-compartment tanker, the tank allocation component is omitted according to the authors because it is not important when looking at the problem from a strategic setting. An integer programming model for both a single ship and for a whole fleet is proposed and solved using a commercial solver. Important problem characteristics such as cargo compatibility and ship stability are not mentioned in this paper but are part of Neo et al. (2006). Here an integer programming model is also the main contribution. Now the model includes the aforementioned constraints on cargo compatibility and ship stability. It is solved using a commercial solver and exhibits very large running times. For a single ship, 10 compartments and 10 potential cargoes, running times are above 18,000 seconds. In the same line of research, Coccola and Mendez (2013) consider basically the same integer programming model as in the previous papers. This approach does not take ship stability and hazmat constraints into consideration. The paper contains the description of an iterative heuristic aimed at situations where excessive running times prohibit optimisation for an entire fleet. The problem is decomposed into single ship problems and cargo conflicts are resolved solving a 2-ship problem in an iterative manner. Authors report a 40% improvement in profit over the manual plan on a single instance and a "significant" improvement over "other" approaches.

Kobayashi and Kubo (2010) consider a tanker problem, though without hazmat and stability constraints. They decompose the problem into a tank allocation problem and a routing problem. Both problems are modeled and solved as set partitioning problems, but for the routing problem the number of columns necessitates column generation. Using column generation to solve the LP relaxation, an integer solution is afterwards found by branch-and-bound on the generated columns. In addition, Wu et al. (2011) describe a decision support system (DSS) that includes the relevant real life constraints and a graphical user interface to deliver a real-life applicable DSS. The optimization is based on a relatively simple heuristic but includes routing and allocation constraints in an integrated fashion. The inclusion of hazmat constraints is, however, not obvious and the method only works by inserting new cargoes to an already existing schedule. Oh and Karimi (2008) describe another DSS. They assume that ship stability can always be achieved by using ballast tanks. They optimise for a fleet of multi-compartment tankers by decomposing the problem into a tank allocation problem and a routing problem. Instead of using column generation as in (Kobayashi and Kubo, 2010) this paper only contains a simple enumerative approach for generating the routes for the set partitioning problem. Cargo allocation is done using a heuristic which is not described in any detail.

In contrast to (Hvattum et al., 2009), the papers mentioned above allow cargoes to be picked up or rejected, which most often implies profit maximisation instead of a feasibility focus. As in (Hvattum et al., 2009), Schaus et al. (2012) assume a fixed route and also focus on feasibility. The paper considers a set of cargoes to be placed on a multi-compartment tanker without any notion of route and therefore loading and unloading of cargoes along the route. The paper does not look at ship stability but does incorporate hazmat constraints, compatibility with previous cargoes, and the sealing on the tanker walls in a constraint programming approach.

Fagerholt and Christiansen (2000) consider a tramp ship problem with adjustable compartments. Here the cargo hold can be partitioned into smaller holds by inserting bulkheads at a discrete number of positions. The problem is a combined routing and allocation problem for a fleet of ships but ship stability and hazmat constraints are not considered. The solution approach is based on an iterative heuristic using a priori generation and dynamic programming to determine the optimal schedules. Each iteration constructs ship routes using a variant of the Traveling Salesman Problem with incorporated allocation, time windows and precedence constraints.

Another important maritime stowage problem is the container stowage problem found in liner shipping. The two problems share some of the same components such as ship stability and spatial separation requirements for dangerous goods. Also, finding a feasible stowage plan is often more important than finding an optimal one. However, there are important operational differences that call for tailor made methods for each of these segments. To mention a few, we note that container stacking calls for efficient cargo handling to minimize container shifting when loading and unloading and this is of course not an aspect in bulk shipping. On the other hand, in the container stowage problem there is no need to consider individual tank constraints, such as e.g. tank capacity and tank coating. For further information on the container stowage problem and solution approaches see e.g. (Martin et al., 1988; Wilson and Roach, 2000).

Transportation in compartments also exists in road transport. Most applications come from distribution of petroleum products; other examples are waste collection and food. In road transport, compartments can be fixed in size or flexible. In (Derigs et al., 2011) a survey and computational comparison of current literature within road transport is presented. Interestingly, road transport always assumes integrated optimization of routing and cargo allocation, and, in addition, the majority of developed approaches are based on heuristics or metaheuristics. Quite often when multiple compartments are considered in the literature, the objective is to just deliver from a central depot to e.g. petrol stations (Cornillier et al., 2009) and not to combine both pickup and delivery as we do here. Furthermore, in these problems bulk ship constraints such as stability are not considered.

The *segregated storage problem* is another related compartment oriented storage optimization problem. In this problem a silo with a number of compartments has to be filled with grain and only one type of grain can go into each compartment. However, contrary to our problem, identical "cargoes" can be mixed into the same compartment. In (Barbucha and Filipowicz, 1997) the segregated storage problem and some variants of the basic problem are compared to storage problems in transport in general but also in particular to maritime transportation. The complexity of the different variants is described and a few numerical examples are given.

8.3 Solution Method

The aim of our work here is to develop an efficient solution method for finding feasible tank allocations in a very short amount of time. Finding a good allocation is left for operational planning where the solution method proposed by Hvattum et al. (2009) is sufficient with respect to time. Our search for a feasible tank allocation assumes that initial crude feasibility tests have been performed, e.g. verifying that the total cargo volume during each voyage leg does not exceed total ship capacity.

Because we focus on feasibility, we can disregard tank cleaning costs. This means that any two cargoes can potentially follow each other in a tank as we can just pay the cost of cleaning. Thereby, we no longer have to keep track of previous tank allocations. Furthermore, if the ship is at some point empty during its route, we can split the full problem into smaller *subproblems* or *subinstances* since any voyage leg prior to the ballast leg will be independent of any voyage leg after the ballast leg.

Even though we disregard cleaning costs, the decisions at any voyage leg still affect decisions at future voyage legs since we are not allowed to move a cargo once it has been allocated. Consequently, allocating cargoes one by one as they are picked up along the route will be a bad idea. We can easily end up allocating a cargo to tanks that are required for a feasible allocation of another cargo picked up later. Our heuristic is therefore based on a priority ordering of cargoes that defines the order in which they will be allocated one by one. The function *selectCargo()* returns the next cargo to allocate. Each time a cargo has been allocated, the chosen tanks are made unavailable for all cargoes onboard with the considered cargo as well as neighboring tanks if there are any conflicting cargoes. The priority ordering for all cargoes affected by the chosen allocation is then updated before selecting the next cargo to allocate.

Ship stability at a voyage leg cannot be calculated before all cargoes onboard during that particular voyage leg have been allocated. Therefore, when allocating cargoes one by one, we initially only use an estimate of ship stability based on allocated capacity rather than on the actual cargo amount allocated to each tank. This means that we initially only reserve sufficient tank capacity for one cargo at a time using a function called *selectTanks()* which takes a specific cargo as its parameter. If we manage to find sufficient capacity for all cargoes, we check if there exists a combination of cargo amounts to tanks, which can secure ship stability during the entire route. This is done by solving a simple linear program (LP) where each cargo is predefined to be allocated to specific tanks, i.e. solving (8.1), (8.2), (8.3), (8.6) and (8.9) with all x variables fixed. The function *solveLP()* performs this task. Note that ballast tanks can easily be incorporated by including them in these LPs.

8.3.1 Diversification

The quality of the heuristic will naturally depend on the quality of the two procedures for selecting the next cargo to allocate and for selecting tanks when reserving capacity for the selected cargo. We have chosen to keep these two selection procedures simple and instead introduce randomness in each of them to produce diverse results with each call to the heuristic. In Section 8.3.2 we describe how randomness is introduced into the cargo selection procedure while in Section 8.3.3 we describe how it is introduced into the tank selection procedure. Within a given time limit, we then allow the heuristic to restart each time it fails to find a feasible allocation for a given subinstance, whether the infeasibility comes from some cargoes not being fully allocated or from instability when determining cargo amounts. Restarting simply means that all cargoes that have already been allocated, are removed from the ship again. The basic outline of the heuristic can be seen in Algorithm 2.

Algorithm 2: Basic heuristic outline

```
1  determine initial cargo priority ordering;
2  split problem to create list, Subs, of smaller subinstances;
3  while Subs ≠ ∅ AND time limit not exceeded do
4      Sub ← first subinstance in list Subs;
5      while no stable allocation found for Sub AND time limit not exceeded do
6          while cargoes in Sub remain to be allocated AND time limit not exceeded do
7              cargo i ← selectCargo();
8              if not enough available tank capacity to allocate cargo i then
9                  if time limit not exceeded then
10                     restart Sub;
11                 else
12                     stop;
13             else
14                 selectTanks( cargo i );
15                 update available tank capacity for affected cargoes;
16                 update priority ordering of unallocated cargoes;
17         if all cargoes in Sub fully allocated then
18             solveLP();
19             if allocation is unstable then
20                 if time limit not exceeded then
21                     restart Sub;
22                 else
23                     stop;
24             else
25                 remove Sub from Subs;
```

8.3.2 Cargo priority - function *selectCargo()*

We prioritise cargoes based on the ratio of the volume of each cargo and the amount of available tank capacity for this cargo. A tank is available if it is both compatible and vacant, also with respect to neighbor cargoes. Obviously, the closer to one this ratio gets, the more important it is to allocate the cargo, since only few of the available tanks can be occupied by other cargoes before it becomes impossible to allocate the given cargo. For a cargo i we denote this estimate of importance by I_i. The basic idea of the cargo selection procedure is then to iteratively select the cargo with the highest value of I_i. Note that if $I_i > 1$, it means that, due to already allocated

cargoes, there is no longer enough available tank capacity left to allocate cargo i. In such situations, we restart the heuristic.

In order to diversify the results of the heuristic, we add some randomness to this otherwise deterministic selection procedure. We do this by allowing the procedure to sometimes discard the most important cargo and instead select a less important cargo. However, if the most important cargo, cargo i, has I_i above a threshold value that we denote I, we always select this cargo. The reason for this threshold approach is that a cargo i with high I_i will only be able to spare few of its available tanks to other cargoes before it becomes impossible to allocate cargo i. Therefore, it will most often be best to allocate cargo i first. The most important cargo, cargo i, with $I_i < I$ will now be selected with a probability of P. We tune the parameters I and P in Section 8.5. Preliminary tests show that the deterministic cargo selection procedure performs best and so, we want to ensure that this approach is followed before adding randomness. Therefore, the first run of the heuristic will use the deterministic selection procedure while the randomised approach will be used after restarting. The basic outline of the cargo selection procedure is then as presented in Algorithm 3 where $\mathcal{C}[0]$ denotes the first element in the list $\mathcal{C}$ and $rand(0,1)$ denotes a random number between 0 and 1.

Algorithm 3: Outline of cargo selection procedure, i.e. function *selectCargo()*

```
1 C ← list of all unallocated cargoes sorted in order of decreasing importance;
2 cargo i ← C[0];
3 if I_i > I or we did not restart then
4     return cargo i;
5 else
6     while C contains at least 2 cargoes and rand(0,1) > P do
7         erase C[0] from C;
8     return C[0];
```

8.3.3 Reserving sufficient capacity - function *selectTanks()*

We allocate each cargo as stable as possible by iteratively choosing tanks that have moment arms in the opposite direction of the ships current stability and strength estimates. Assume for example that the current condition of the ship is that its stability and strength measures are positive for roll, negative for trim and negative for strength. Then we only consider available tanks with negative moment arm for roll and positive moment arm for both trim and strength. Naturally, if no such tanks remain, we look for tanks that fulfill two out of these three moment characteristics and we continue this process of lowering our requirements until we find available tanks. During preprocessing, we therefore group the tanks according to their moment arms for the various stability and strength dimensions. E.g. considering roll, trim and strength, we obtain 27 such *stability dimension groups* (SDG) since a moment arm can also be neutral, i.e. equal to zero. To determine the correct SDG to consider, we have a function called $selectSDG()$ that, given the current condition of the ship, tells us which SDG to consider. Notice that multiple SDGs can be eligible if one of the stability and strength measures is currently neutral or if a preferred SDG does not currently contain any available tanks. In such cases we break ties arbitrarily. Once a tank has been selected, we can determine if further capacity is required to fully allocate the cargo, and if there is, we update the stability and strength measures to determine a new SDG of eligible tanks to choose from.

In principle each cargo does not by itself have to be allocated in a stable manner since cargoes can outbalance each other. However, many cargoes are at some point onboard the ship alone ("alone-cargoes") and must therefore be allocated in a stable manner. Thereby, any cargo that is at some point onboard the ship with only an alone-cargo, must also be allocated in a somewhat stable manner. This reasoning can continue, thus motivating the search for stable allocations for each individual cargo. Furthermore, allocating each cargo in a stable manner yields robust plans since the stability of the ship does not rely on cargoes to outbalance each other. Hence, any cargo can in principle be removed from the route without causing instability. Note however, that we do

not *require* each cargo to be allocated in a stable manner on its own but simply *seek* to find a stable allocation for each cargo. Often it is not possible to find stable allocations for the individual cargoes and furthermore, the added randomness in the tank selection procedure can easily cause us to reject them if they exist.

Finding a feasible allocation for all cargoes relies on a clever allocation of each individual cargo as the cargoes are in effect competing for the same tanks. For cargoes that are not in conflict with any other cargoes, a clever allocation mainly relates to the capacity utilisation of the chosen tanks. For cargoes in conflict with other cargoes, it is however equally important to try to confine the cargo to smaller groups of tanks as opposed to scattered all over the ship whereby a lot of neighboring tank capacity would be unavailable for cargoes in conflict with this cargo. Therefore, we distinguish between two types of cargoes:

> A cargo i is a *conflict cargo* if there is at least one other cargo j that is in conflict with cargo i and where cargo j has not yet been allocated.
>
> A cargo i is a *non-conflict cargo* if it is not in conflict with any other cargoes or if all conflicting cargoes have already been allocated.

When allocating a non-conflict cargo, we then focus on capacity utilisation and simply sort the compatible and vacant tanks within each SDG by size in decreasing order. When allocating a conflict cargo we instead sort the compatible and vacant tanks within each SDG by decreasing ID, which reflects their location on the ship. This way we will choose tanks grouped together although we might use several groups of tanks in different locations of the ship to secure stability and strength of the ship.

In Figure 8.1 we give a very small example of how the ID's reflect the location of the tanks onboard the ship. Note that with this very crude estimate of neighboring tanks and groupings, our heuristic will correctly assume that tanks 1 and 3 are both neighbors to tank 2 but it will have no idea that tank 5 is also a neighbor to tank 2. Furthermore, it will assume that tanks 3 and 4 are neighbors, which clearly they are not. This sort of ship specific information could obviously be used to improve the tank sorting function. However, since the tanks are sorted within each specific SDG, the tank sorting function does not reflect neighboring tanks across different SDGs. Thereby, the effect of improving the tank sorting function might be limited and furthermore it would sacrifice simplicity and make the heuristic much less generic than when we do not use the actual layout of the considered ship. Therefore, we have chosen to use this very crude estimate of neighboring tanks.

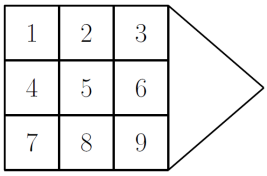


Figure 8.1: Example of tank IDs to reflect ship layout

For all cargoes we use a capacity utilisation threshold so that no tank can be selected if the resulting allocation has a capacity utilisation less than this threshold. However, we allow this threshold to vary with the cargo since cargoes onboard during crowded voyage legs must utilise capacity better than cargoes only onboard during less crowded voyage legs. We therefore determine the minimal capacity utilisation of each voyage leg by dividing the total onboard cargo volume by ship capacity. For each cargo, we then consider all the voyage legs that the cargo is onboard the ship during, and the maximal of the minimal capacity utilisations corresponding to these specific voyage legs, is then the capacity utilisation threshold for this particular cargo. We denote the capacity utilisation threshold for cargo i by U_i.

If we want to generate solutions with high capacity utilisation, we can simply exchange this cargo specific threshold by a sufficiently high uniform threshold used for all cargoes. This approach will utilise ship capacity much better than when using cargo specific thresholds and thereby also

better preserve flexibility to accommodate future cargoes. However, it will increase the running time of the heuristic and since our main concern is feasibility, we use the cargo specific thresholds.

Naturally, we might have to lower the capacity utilisation threshold gradually if no tanks remain that fulfill the utilisation requirement. In cases where multiple SDGs are applicable, we only lower the utilisation threshold if none of the applicable SDGs contain an eligible tank. For conflict cargoes, choosing groups of tanks is very important and as soon as we start discarding tanks in the otherwise location based ordered tank list, we are in effect choosing tanks that are scattered all over the ship. Therefore, gradually decreasing the capacity utilisation requirement with only small decrements does not seem a good idea for conflict cargoes. As soon as we discard a tank due to capacity utilisation, to avoid discarding many tanks, we should instead lower the utilisation requirement almost to a point that it becomes obsolete. For non-conflict cargoes this, however, is not necessary. Therefore, we use separate parameters for the two cargo groups for the incremental lowering of the capacity utilisation threshold when we run out of tanks. For non-conflict cargoes and conflict cargoes we respectively denote these parameters ΔU_{NC} and ΔU_C and tune them in Section 8.5.

To ensure diversification we add elements of randomness to the otherwise deterministic tank selection procedure. Similar to the cargo selection procedure, we do this by only accepting a given tank with a certain probability. To keep the parameter list as short as possible, we reuse the probability parameter P from the cargo selection procedure. For the conflict cargoes, we again need to ensure that we choose tanks in small groups rather than tanks scattered all over the ship. Therefore, for conflict cargoes we use an additional parameter, ΔP, that is used to iteratively increment the tank acceptance probability each time a tank has been discarded. Thereby, when a tank is discarded, we lower the probability of discarding the next tank in the list. All discarded tanks are stored in case we run out of available tanks and must return to these. The discarded tanks are also grouped according to their stability and strength dimensions and further sorted within these groups according to either size or ID, depending on the cargo type. We tune the parameter ΔP in Section 8.5.

Preliminary tests show that for conflict cargoes it is best to use the deterministic tank selection procedure prior to restarting and the randomised selection approach afterwards and so, we use this approach. This confirms that the deterministic selection approach best enforces the group wise selection criteria. However, for non-conflict cargoes preliminary tests do not indicate any benefits from using the deterministic procedure. For non-conflict cargoes, we therefore always use the randomised approach.

Each restart requires all allocated cargoes to be reallocated and this is obviously much more time consuming than just restarting for a single cargo, i.e. removing this one cargo from the ship and reallocating it. Therefore, iteratively allocating the cargoes and then restarting the whole process in case of infeasibility is not an attractive approach. Instead, each time we allocate a cargo, we iteratively reallocate it until the quality of the allocation is sufficiently high. Keeping the strict computation times in mind, we use a very crude estimate of quality by simply requiring the capacity utilisation of an allocation for cargo i to be at least as high as the capacity utilisation of the most crowded voyage leg, that cargo i is onboard during, i.e. at least as high as U_i. We use the parameter L to limit the number of allowed reallocations of each cargo. If we reallocate as many times as allowed without finding a good enough allocation, we use the best one found so far. The parameter L is also tuned in Section 8.5.

Since the tank selection procedure for conflict cargoes is deterministic before restarting, there will be no point in reallocating conflict cargoes before restarting. After restarting, conflict cargoes are reallocated, just as non-conflict cargoes always are. However, preliminary tests also show that after restarting it can be beneficial to allow the first iteration of the reallocation procedure for conflict cargoes to follow the deterministic tank selection procedure. Reusing the probability parameter P, we then only perform the first iteration of the reallocation procedure for conflict cargoes, i.e. the deterministic one, if a randomly generated number between 0 and 1, $rand(0, 1)$, is less than P.

Algorithm 4 shows the basic outline of the tank selection procedure for a generic cargo i. Here, ΔU_i is equal to ΔU_C if cargo i is a conflict cargo, and equal to ΔU_{NC} if cargo i is a non-conflict cargo.

Algorithm 4: Algorithm for selecting tanks for cargo i, i.e. function *selectTanks(cargo i)*

```
if cargo i is non-conflict OR we have restarted then
    if cargo i is non-conflict OR rand(0,1) < P then
        count ← 0;
    else
        count ← 1 (i.e. skip deterministic iteration);
else
    count ← L (i.e. don't reallocate);
while count ≤ L do
    create size/ID sorted list of compatible and vacant tanks for each SDG;
    u ← U_i;
    stability estimate e ← 0;
    clear allocation and discarded tank list from last iteration;
    p ← P;
    while further capacity required do
        G ← selectSDG(e);
        tank t ← first tank in sorted tank list of G;
        while tank t ≠ NULL, i.e. tanks remain do
            if capacity utilisation ≥ u then
                remove tank t from list of tanks from G;
                if not using discarded tanks AND (cargo i is non-conflict OR (using
                reallocation AND count > 0)) then
                    num ← rand(0,1);
                else
                    num ← 0 (i.e. no tanks are discarded);
                if num < p then
                    allocate cargo i to tank t;
                    update e using capacity of tank t;
                    tank t ← NULL, i.e. end loop to allow switching to new SDG;
                else
                    add tank t to size/ID sorted list of discarded tanks for G;
                    tank t ← next tank in sorted tank list of G;
                    if cargo i is conflict cargo then
                        p ← p + ΔP;
            else
                tank t ← next tank in sorted tank list of G;
        if no tank was found AND no other SDGs are applicable then
            if tanks remain that are not discarded then
                u ← u − ΔU_i;
            else
                from discarded tanks, create size/ID sorted tank list for each SDG;
                u ← U_i;
    if using reallocation then
        if capacity utilisation ≥ U_i then
            count ← L, i.e. exit reallocation loop;
        else
            update best allocation found so far;
    count++;
if using reallocation AND reallocated as many times as allowed then
    use best allocation found;
```

8.4 Data

In (Hvattum et al., 2009), an instance generator is developed to create a varied set of realistic and feasible TAP instances based on two real tank ships. The instance generator is used to generate 720 data instances and we base our computational study on these instances. Here, we briefly describe the main features of the data instances, while a thorough description of the instance generator can be found in (Hvattum et al., 2009).

There are two tank ships of different size and configuration. The smaller ship has 24 tanks and can carry up to 10,000m^3 while the larger ship has 38 tanks and a capacity of 45,000m^3. There are two types of tanks, namely stainless steel tanks and tanks with a zinc silicate coating. The data instances only contain data for stability and strength restrictions with respect to roll. This means that, although our heuristic is generic enough to handle several stability and strength dimensions, we will here only consider the roll dimension and therefore only have three stability and strength dimension groups, containing, respectively, tanks with negative, neutral and positive moment arm for roll. There are three categories for cargoes:

1. cargoes that can go into any tank and do not conflict with any other cargoes
2. cargoes that can go into any tank but which conflict with all category 3 cargoes
3. cargoes that can only go into tanks with zinc silicate coating and which conflict with all other cargoes in category 2 and 3

Cargoes that are in conflict with each other cannot simultaneously occupy tanks that share a side, i.e tanks that are neighbors.

In each data instance the first ten generated cargoes are defined to be locked, i.e. they are already allocated and cannot be moved. They simply act as the history of the ship and cause some tanks to be occupied when planning starts. Hvattum et al. (2009) generate instances with, respectively, 20, 30 and 40 cargoes where the first ten are locked cargoes. To create a varied set of instances, they use three additional parameters:

D denoting the probability distributions for the category of each load. They allow four settings for this parameter and label them $D1(0.6, 0.4, 0.0)$, $D2(0.5, 0.4, 0.1)$, $D3(0.4, 0.4, 0.2)$ and $D4(0.3, 0.4, 0.3)$. Here $D1$ refers to the case where 60% of the cargoes are expected to come from category 1, 40% from category 2 while no cargoes are expected from category 3.

F denoting the minimum/maximum capacity utilisation of the ship before loading/unloading. This parameter is allowed three settings which are labeled $F1(0.65, 0.35)$, $F2(0.75, 0.25)$ and $F3(0.85, 0.15)$. Here $F1$ refers to the case where the ship will visit pickup locations until the the total load exceeds 65% of ship capacity and then start to visit discharge locations until the total load becomes less than 35% whereafter the ship will again visit pickup locations.

V denoting the distribution of load volumes. This parameter is allowed three settings labeled $V3(1000 - 5000m^3)$, $V6(3000 - 9000m^3)$ and $V12(8000 - 16,000m^3)$. Here $V1$ denotes the case where cargo sizes follow a uniform distribution over the interval 1000 to 5000m^3.

Combining ship type, number of cargoes and the above three parameters while ruling out unrealistic combinations, Hvattum et al. (2009) obtain 144 parameter setting groups. For each of these 144 groups, they randomly generate five feasible instances yielding a total of 720 instances for which the existence of feasible solutions is verified. We let TAP_T24C20D3F1V3_01 denote the first generated instance for the 24 tanks ship with 20 cargoes (including ten locked cargoes) and with parameters D, F and V set to, respectively, 3, 1 and 3.

8.4.1 Instability for locked cargoes

Since solving any one of the above 720 TAP instances only entails finding a feasible allocation for the unlocked cargoes, the feature in the instance generator that verifies the existence of a feasible solution, does not check the allocation of the locked part of the instance. This unfortunately means that for quite a few of the 720 instances, the ship is at some point unstable during the locked part

of the ship route. Solving the problem with a mixed integer formulation and CPLEX, as done in (Hvattum et al., 2009), this instability is not a problem since the entire solution space is explored, including solutions where one cargo can be allocated solely to one end of the ship in order to outbalance a locked cargo placed in the opposite end of the ship. However, since our heuristic tries to allocate each individual cargo in a stable manner, it will not perform well on instances with built-in instabilities, i.e. instances where the locked part contains cargoes or groups of cargoes that are placed in an unstable manner. Note that even though the heuristic *seeks* stable allocations for each cargo, it *can* allocate cargoes in an unstable manner and thereby explore the entire solution space. However, the chances of the heuristic finding extreme solutions are slim and thereby we are penalised for an instability created before the ship was even handed over to us. If our heuristic were used continuously, then all cargoes previously allocated, i.e. locked cargoes, would also be allocated in a somewhat stable manner and then this would not be a problem. Therefore, a computational study using these instances can, in this respect, be expected to yield a conservative estimate of the success rate of our heuristic. In order to get a feel for the effect of excluding such built-in instabilities, we consider three distinct sets of data instances during our computational study:

Set_{720} Contains all 720 instances

Set_{648} Contains the 648 instances that are stable when the ship is "handed over" to us, i.e. just before picking up cargo number 11

Set_{486} Contains the 486 instances that are stable during the entire locked part of the last subinstance before the ship is "handed over" to us, i.e. during the first subinstance that we have to solve. Note that we do not require the ship to be stable during the entire locked part of the route as we do not care about what happens before the first subinstance that we have to solve. I.e. if the ship is unstable and then becomes empty before allocating other locked cargoes that we have to take into account, then we do not care about the instability

To properly understand the reason for including Set_{486}, consider the example illustrated in Table 8.1. Here, 'Action' refers to the port action taken at a particular point in time and time progresses as we move to the right in Table 8.1. 'Onboard' refers to the cargoes onboard immediately after this action while 'Roll' refers to the roll measure of the ship immediately after the action. We let $-K$ and K denote, respectively, the lower and upper threshold for stability in the roll dimension. For port actions, '9+' and '9-' refer to the actions of respectively loading and discharging cargo number 9, and similarly for higher numbers. We assume that the ship becomes empty just before loading cargo 9 whereafter cargo 9, 10, 11, and 12 are all picked up and then again discharged in the same order as they were picked up. This means that the first subinstance that we have to solve, contains the locked cargoes, cargoes 9 and 10, and the two unlocked cargoes, cargoes 11 and 12, whereby we can confine our example to include just the small part of the ship route given in Table 8.1. Now assume that cargo 9 is allocated in an unstable manner and that cargo 10 is allocated in a similarly unstable manner but that these two cargoes outbalance each other so that the ship is stable when it is handed over to us, i.e. just before picking up cargo 11. So, assume that the allocation of cargo 9 creates a roll measure of $-K-1$ and that the allocation of cargo 10 creates a roll measure of $K+1$. The roll measure will then be 0 when the ship is handed over to us. Then our heuristic will try to allocate cargoes 11 and 12 in a stable manner and we can, for argument's sake, assume that the effect of their allocations on the roll measure is zero. After the pickup of cargoes 11 and 12, we will discharge cargo 9 and thereby obtain a roll measure of $K+1 > K$. This means that even though the ship was stable when handed over to us, and we

Table 8.1: Small example to illustrate reason for considering Set_{486}

Action	9+	10+	11+	12+	9-	10-	11-	12-
Onboard	9	9,10	9,10,11	9,10,11,12	10,11,12	11,12	12	-
Roll	$-K-1$	0	0	0	$K+1$	0	0	0

managed to allocate both cargo 11 and 12 in a fully stable manner, we ended up with an unstable ship. Theoretically, the heuristic could find a feasible solution to this problem. However, since the heuristic seeks to allocate each individual cargo in as stable a manner as possible, chances are that it will not find a feasible solution. Thereby, we are penalised for an instability created before the ship was even handed over to us. Assuming that the heuristic was used continuously, such instabilities would never occur and generally, we do not anticipate such instabilities in real life. Therefore, we find it reasonable to include Set_{486} in our computational study. Note however that even when confining our computational study to the 486 instances, there can still be instances that will affect the success rate of our heuristic in a negative manner compared to if our heuristic had been used continuously. Assume for example that we extend the subinstance in the example from above to include the locked cargo 8 and that cargo 8 and 9 are now both allocated in a manner that affects the roll measure with $-\frac{1}{2}K$, i.e. after picking up cargo 9, the ship will have a roll measure of $-K$ which just leaves the ship stable. Adding cargo 10 we obtain a roll measure of 1 which is again stable. Adding cargoes 11 and 12 maintains the roll measure at 1. However, once we discharge cargoes 8 and 9, we obtain a roll measure of $K+1$ and thereby an unstable ship even though the locked part contained no instabilities and we managed to allocate the unlocked cargoes in a fully stable manner. However, since such instances are per definition feasible and we do not want to exclude too many instances, we limit our computational study to the three sets above and simply note that the success rate of our heuristic on these sets can be expected to be a conservative estimate of the success rate that could be achieved if the heuristic was used continuously.

8.5 Tuning

Our heuristic has the following parameters, that must be tuned:

- P: probability of accepting a chosen cargo or tank or using the deterministic tank allocation for conflict cargoes
- I: threshold value for cargo importance that eliminates randomness in the cargo selection procedure
- ΔU_{NC} and ΔU_C: reduction parameter for non-conflict and conflict cargoes respectively. Used to iteratively lower the capacity utilisation threshold
- ΔP: increment parameter used to iteratively increase the probability of accepting a chosen tank for conflict cargoes
- L: number of times the heuristic is allowed to randomly reallocate a cargo

As discussed in Section 8.4, results from running our heuristic on the reduced data set Set_{486} yields the most realistic estimate of its performance compared to data sets Set_{720} and Set_{648} containing built-in instabilities. Therefore, during tuning we only consider the instances in Set_{486}. For the larger ship with 38 tanks, this set only contains 134 instances and these are again spread over a wide variety of instance types with varying cargo count, cargo sizes etc. As mentioned in Section 8.4, only five instances of each type are generated and with the removal of instances with built-in instabilities, this number is significantly lower for most instance types. Thereby, it will be difficult to extract only a subset of instances for tuning while at the same time retaining a diverse and representative sample of instances. Therefore, we have chosen to tune on all instances in Set_{486}; however, this will bias our conclusions slightly when performing the computational study on this data set in Section 8.6.

We tune two versions of the heuristic: One for the smaller ship with 24 tanks (denoted T24) and one for the larger ship with 38 tanks (denoted T38). During development of the heuristic and preliminary testing, we obtained qualified estimates for the initial parameter test value ranges. Within the initial interval for each parameter we chose three test values. With six parameters, each allowed three different values, we obtained $3^6 = 729$ parameter scenarios. Due to the randomness of the heuristic, we ran each scenario five times on each data instance in Set_{486} and evaluated the average performance of each parameter scenario. This resulted in a total of $5 \cdot 729 = 3645$ scenarios to be run on 486 instances. Since the heuristic is developed to efficiently solve numerous

subproblems in a tactical setting, we allowed a maximum run time of 0.2 seconds. On average, each scenario runs through all 352 data instances for T24 in 3.3 seconds and 1.5 seconds for all 134 data instances for T38. Therefore, choosing the parameter setting with the best average performance from each of the six initial lists of test values for each ship took a total of $(3.3+1.5)\cdot 3645 = 17,496$ seconds, i.e. under 5 hours. The best choice of each parameter value allowed us to further reduce the initial test value ranges and repeat the process iteratively until the optimal parameter values were determined after three iterations. I.e. assuming that the initial test values for a parameter were $\{0.4, 0.6, 0.8\}$ and we concluded that 0.6 was the best of these, then in the next iteration we narrowed the range and instead tested $\{0.5, 0.6, 0.7\}$. A few times the parameter setting with best average performance was on the boundary of the test value range and so we re-expanded before again zooming in on the optimal parameter setting.

We encountered ties between multiple parameter scenarios several times during the tuning process. Since the average running time did not vary much between the different parameter scenarios, we broke ties by selecting the scenario with the most stable performance using the standard deviation of the solve count for the five sample runs as a measure of stability. The initial test values as well as the final chosen parameter settings are shown in Table 8.2 for both T24 and T38.

Table 8.2: Tuning results

	Initial test values	T24 Optimal value	T38 Optimal value
P	{0.4, 0.6, 0.8}	0.50	0.60
I	{0.4, 0.6, 0.8}	0.70	0.60
ΔU_{NC}	{0.05, 0.25, 0.45}	0.10	0.60
ΔU_C	{0.3, 0.5, 0.7}	0.45	0.45
ΔP	{0.05, 0.25, 0.45}	0.05	0.15
L	{20, 40, 60}	50	35

From the optimal parameter settings in Table 8.2, we note that P is between 50% and 60% meaning that the heuristic relies on a great deal of randomisation. Finally, we note that using the worst setting from the above test ranges resulted in a success rate that was only 1% lower than when using the optimal parameter setting. Therefore, the heuristic is quite robust to parameter changes.

8.6 Computational Results

In this section we evaluate the performance of our heuristic. We compare our results to the ones obtained in (Hvattum et al., 2009) from using an optimal method and to results from both updating and modifying their method. We also combine our heuristic with the both updated and modified version of the optimal method and evaluate the performance of this hybrid method.

8.6.1 Results from optimal method

First, we briefly summarise the findings Hvattum et al. (2009) obtained solving their mixed integer program directly with CPLEX v.11 on an Intel 2.66GHz processor. In Table 8.3 we summarise their results for three different objective functions. The first objective function is to simply minimize 0, i.e. focus on feasibility, the second seeks to minimise tank cleaning while the last one maximises average capacity of vacant tanks during the ship's route. They allowed a maximum run time of 600 seconds to solve each instance in Set_{720}. To capture the variance in running times, Table 8.3 shows the number of instances for which a feasible solution is found within a given time limit stated in the top row as well as the average running time given in the last column.

On average, the method that focuses on feasibility, i.e. objective (1), is fastest. However, allowing running times of 10 seconds or more, the two other objective functions result in a greater number of feasible solutions. For all three methods, the average running time is too long for a

Table 8.3: Results from Hvattum et al. (2009)

Objective	Data set	≤ 1	≤ 10	≤ 100	≤ 600	Av. secs
(1) feasibility	Set_{720}	609	662	680	683	2.5
(2) min tank cleaning	Set_{720}	555	677	707	716	4.1
(3) max av. vacant cap.	Set_{720}	546	672	711	717	5.1

method that is to be used to solve a subproblem numerous times in e.g. column generation or local search.

In order to enable a fair comparison with the results from our heuristic, we ran their algorithm with the three different objectives on the same machine used for our heuristic results and also used the newer version 12.4 of CPLEX, that we use for solving the small LPs when determining the cargo amount to put into each tank. Table 8.4 therefore gives the same numbers as Table 8.3, however, for a 4.0GB RAM PC with Intel Core2 Duo, 2.4 GHz processor. Since this updated version is faster than the original version, we have added a column with the smaller time limit of 0.1 second and a column with time limit 250 seconds.

Table 8.4: Results from updated version of optimal method from Hvattum et al. (2009)

Objective	Data set	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	≤ 600	Av. secs
(1) feasibility	Set_{720}	298	659	715	719	720	720	0.767
(2) min tank cleaning	Set_{720}	205	557	694	719	720	720	1.822
(3) max av. vacant cap.	Set_{720}	23	525	690	715	717	717	4.912

From Table 8.4 we see that the updated version of the algorithm is both faster and better than the old one, as it is now able to solve all instances when using objective (1) or (2). These two methods actually solve all instances within 250 seconds while using objective (3) fails to solve all instances within the time limit of 600 seconds. On average, the no objective method is still fastest. However, now the two other objective functions do not yield better results after 10 seconds as they did before. Instead, using no objective is now best on all accounts regardless of the time limit. Therefore, when comparing with our heuristic, we use the no objective method and denote this updated optimality-based method by 'UpdOpt'. We also run this version of the algorithm on the two sets Set_{648} and Set_{486} to enable comparison with our heuristic on these sets. The results from these runs are presented in Table 8.5 where the time limit has been reduced from 600 to 250 seconds due to the speed up of the algorithm.

Table 8.5: Results from UpdOpt method

Objective	Data Set	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
(1) feasibility	Set_{648}	240	598	644	648	648	0.463
(1) feasibility	Set_{486}	196	463	483	486	486	0.374

From Tables 8.4 and 8.5 we note that removing instances with built-in instabilities significantly reduces problem complexity and thereby running times. However, if our method is to apply to bulk operators of all sizes, the running time of the algorithm is still too long for the method to be applicable as a subproblem solver in a column generation or local search based framework.

8.6.2 Modifying the optimality-based method

As already mentioned, when the focus is as here on feasibility rather than optimality, there is no longer any need to include the cost of tank cleaning nor the constraints for cleaning as we can simply assume that tanks are cleaned if necessary, regardless of cost. Removing the cleaning constraints (8.7) from the model (8.1)-(8.10) means that we can further remove the binary cleaning variables, z_{ct}, to further reduce the model size and thereby also the running times. However, for some reason this observation is neither mentioned nor explored by Hvattum et al. (2009). Instead, we have modified their algorithm by removing all cleaning variables and constraints and denote this updated and modified optimality-based method by 'ModOpt'. Table 8.6 shows the improved results from running this version of the algorithm on the three data sets.

Table 8.6: Results from ModOpt method

Objective	Data Set	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
(1) feasibility	Set_{720}	310	665	716	719	720	0.726
(1) feasibility	Set_{648}	293	605	645	648	648	0.402
(1) feasibility	Set_{486}	263	467	483	486	486	0.282

From Table 8.6 we see that removing all variables and constraints related to cleaning, reduced the running times by 5-25%. Even so, these running times are still a bit too long for our purpose.

8.6.3 Results from the developed heuristic

Now we are ready to present the results from the heuristic described in Section 8.3. As mentioned in Section 8.6.1, all heuristic tests have been run on a 4.0GB RAM PC with Intel Core2 Duo, 2.4 GHz processor. In Table 8.7 we show the summarised results from running the heuristic once on each of the three different data sets using the optimal parameter setting derived in Section 8.5. Note that for completeness we here include tests on the data sets Set_{720} and Set_{648} even though we know that the heuristic will not perform well on these due to the built-in instabilities in these data instances. Even though the heuristic is developed to quickly assess feasibility and that it has therefore been tuned with a time limit of only 0.2 seconds, we here allow a time usage of 250 seconds to be able to compare with the results from Hvattum et al. (2009). Later, we will lower this time limit and run multiple tests to investigate the stability of the heuristic.

Table 8.7: Results from a single run of the heuristic

Data Set	≤ 0.01	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
Set_{720}	289	671	702	712	715	715	1.889
Set_{648}	366	623	637	641	644	644	1.776
Set_{486}	290	475	482	484	485	486	0.426

From Table 8.7 we see that the heuristic is able to solve all instances in Set_{486} and solves 99% of the instances within 1 second. In fact, 99% of the instances are solved already within 0.4 seconds. As expected, the heuristic performs worse on Set_{648} and Set_{720}. The average running time on these sets are relatively high and the instances with built-in instabilities obviously play a major role here as any unsolved instance causes a time usage of 250 seconds whereby the average solve time is significantly higher than without these instances. Comparing our results with the ones in Tables 8.4 and 8.5, we see that our average running times are longer than the ones from the

UpdOpt and ModOpt methods. However, these averages are, as just mentioned, highly affected by just a few difficult instances. Looking instead at the distribution of time usage, we see that our heuristic performs much better than both the UpdOpt method and the ModOpt method when running time is limited. In fact, the time limit has to be at least 10 seconds for the UpdOpt method and the ModOpt method to match or outperform our heuristic.

To get an idea of which instance groups are complicated, Table 8.8 shows more detailed information on the heuristic results for Set_{486}. Looking just at the average running time can be very misleading since a single instance with a long running time will have a huge impact on the average running time. Therefore, when determining which instance groups are complicated it is more accurate to consider the distribution of running times and consider for which groups the majority of the instances are solved quickly. To assist in the understanding of Table 8.8, for each instance group we put a '-' in entries where all instances in the group have already been solved at a lower time limit. Thereby, an easy rule of thumb becomes: the more '-' entries in a row, the easier the instance group corresponding to this row was to solve.

Table 8.8: Detailed results from heuristic on Set_{486}

Subset	#INST	≤ 0.01	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
TAP	486	290	475	482	484	485	486	0.426
C20	161	153	160	161	-	-	-	0.012
C30	168	95	164	165	166	167	168	1.191
C40	157	42	151	156	157	-	-	0.031
D1	114	64	113	114	-	-	-	0.011
D2	116	74	115	116	-	-	-	0.013
D3	124	76	119	123	124	-	-	0.060
D4	132	76	128	129	130	131	132	1.490
F1	136	79	132	135	136	-	-	0.050
F2	165	94	160	163	164	164	165	1.005
F3	185	117	183	184	184	185	-	0.185
T24/V3	172	149	169	171	172	-	-	0.024
T24/V6	180	63	180	-	-	-	-	0.012
T38/V6	41	24	33	38	39	40	41	4.867
T38/V12	93	54	93	-	-	-	-	0.012

From Table 8.8 we first note that, rather surprisingly, the cargo count does not give any indication of problem complexity for our heuristic. Instead, we can use the probability distribution for the cargo categories as a clear guide to problem complexity. This makes sense, since the higher the value of D, the more category 3 cargoes there are, and hence, the more conflicts there are, resulting in unutilised neighboring capacity. With regards to the minimum/maximum capacity utilisation, i.e. the F parameter, we see no pattern with respect to complexity. This actually makes sense, since a lower value for F means that the ship is not very crowded, i.e. requirements for capacity utilisation are low, but on the other hand the ship is rarely emptied completely and so we cannot decompose into smaller and easier subproblems. There is a weak trend suggesting that instances for the T38 ship are more difficult than instances for the T24 ship. However, a much more important factor is the cargo sizes. The smaller they are, the more cargoes will be on board simultaneously, and hence, the more complex the combinatorial puzzle will be.

We also note from analysing the unsolved instances, that the majority of these derive from an inability to reserve sufficient capacity for each cargo as opposed to ensuring ship stability. In fact, instability issues account for less than 1% of the unsolved instances, suggesting that the procedure for creating stable allocations when selecting tanks, works very well. Since this method is generic enough to handle instances with more than one stability dimension, i.e. consider other dimensions than roll, this also suggests that the heuristic will work well on instances with other stability dimensions included.

Allowing long enough running times, the heuristic can solve all instances in Set_{486}. However, the heuristic is neither developed for nor tuned to long running times. Therefore, we want to get an idea of the randomness of the heuristic performance with smaller time limits. To explore this, we have run the heuristic five times on Set_{486} with a time limit of 0.2 seconds. In Table 8.9 we report the minimum, average and maximum number of solved instances over the five runs. We also state the standard deviation of the number of solved instances. Note though, that without knowing the true probability distribution, this is a sample deviation assuming equal probability for all outcomes. Therefore, this number is most likely higher than the actual standard deviation where outliers would contribute with less weight. Even so, we note from Table 8.9 that overall performance of the heuristic is quite stable.

Table 8.9: Analysis of heuristic performance over five runs with time limit 0.2 seconds

Data Set	Minimum	Average	Maximum	Std.Dev.
Set_{486}	478	480	481	1.1

8.6.4 Devising a hybrid solution method

So far we have improved the method from Hvattum et al. (2009) and created a heuristic solution method and presented results for both these methods. When comparing these results we see that the heuristic is much faster than the ModOpt method on the majority of instances. However, the heuristic struggles with just a few instances which results in an overall average running time slightly higher than that of the ModOpt method. In Table 8.10 we compare these two methods for running times below 5 seconds.

Table 8.10: Comparing the ModOpt method with the heuristic on Set_{486} with time limit 5 seconds

Algorithm	≤ 0.01	≤ 0.025	≤ 0.05	≤ 0.1	≤ 0.5	≤ 1	≤ 5	Av. secs
ModOpt	0	0	107	263	447	467	482	0.229
Heuristic	290	463	471	475	481	482	484	0.048

From Table 8.10 it is clear that the heuristic is much faster than the ModOpt method on the vast majority of instances. A thorough analysis of the results from these two methods shows that they complement each other quite nicely as the two instances that cause the heuristic trouble are relatively easily solved by the ModOpt method while this instead struggles on lots of instances that are easily solved by the heuristic. This suggests combining the two methods to obtain an even faster method. Since the heuristic is much faster than the ModOpt method on most instances, we combine the two methods by first running the heuristic for a short amount of time and if no feasible solution is found, we run the ModOpt method. However, as described in Section 8.3, the heuristic splits each problem into smaller subinstances and iteratively tries to solve each of these. This means that even if the heuristic failed to find a feasible solution for a given problem instance, it can easily be the case that it managed to find feasible solutions to one or more of the smaller subinstances. We can utilise this fact to enhance the hybrid method even further. Therefore, we combine the two methods by first running the heuristic for a short amount of time and if no feasible solution is found, we run the ModOpt method, however, only on those subinstances that the heuristic failed to solve. Since we only apply the ModOpt method to smaller subinstances, the computation times will obviously be much shorter than what we saw for the full instances. Table 8.11 shows the results from running this hybrid method on the three data sets with a heuristic time limit of 0.2 seconds as the heuristic is tuned for.

Table 8.11: Results from hybrid method

Data Set	≤ 0.01	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
Set_{720}	376	671	701	718	719	720	0.367
Set_{648}	283	616	633	647	648	648	0.123
Set_{486}	295	479	483	486	486	486	0.027

Just as we found from the heuristic test results, Table 8.11 confirms that the removal of instances with built-in instabilities significantly reduces average running time. In fact, going from Set_{720} to Set_{648} causes a 66% reduction in average running time, while going from Set_{648} to Set_{486} yields a 78% reduction. Furthermore, we note from Table 8.11 that Set_{720} requires a time limit of 250 seconds to solve all instances while Set_{648} requires 100 seconds, and finally, all instances in Set_{486} can be solved within only 10 seconds. A closer look at our results shows that these numbers are in fact 185 seconds, 21 seconds and 3 seconds, respectively. We note that with an average running time of just 0.027, this hybrid method is certainly efficient enough to be used as a subproblem solver. In the next section we compare this hybrid method to the previously presented solution methods.

8.6.5 Comparing algorithms

In order to ease comparison between the different methods, we have gathered the summarised results for Set_{486} in Table 8.12.

Table 8.12: Comparing the different algorithms on Set_{486} with time limit 250 seconds

Algorithm	≤ 0.01	≤ 0.05	≤ 0.1	≤ 1	≤ 5	≤ 10	≤ 100	≤ 250	Av. secs
UpdOpt	0	87	196	463	481	483	486	486	0.374
ModOpt	0	107	263	467	482	483	486	486	0.282
Heuristic	290	471	475	482	484	484	485	486	0.426
Hybrid	295	475	479	483	486	486	486	486	0.027

By construction, the heuristic and the hybrid method will perform similarly for the first 0.2 seconds. Thereafter, the hybrid method will utilise CPLEX to solve the remaining instances causing a 94% reduction in average running time compared to the heuristic that struggles with two instances which have a huge impact on the average running time. Note that if the heuristic had been used continuously so that all locked loads were also placed in a stable manner, the heuristic performance would most likely improve. Even so, the heuristic clearly outperforms the two optimality based methods as long as running time is below 10 seconds. After 10 seconds the two optimality based methods are better at solving the last two instances that cause the heuristic trouble. Thereby, their average running times are better than that for the heuristic. Since the hybrid method combines the best of the heuristic and the optimality based methods, it outperforms them all. In fact, no matter the allowed time limit, no other method solves more instances than the hybrid method and its average running time is as much as 93% lower than the updated version of the method originally presented by Hvattum et al. (2009) and 90% lower than our modified version of this method.

8.7 Concluding Remarks

In this paper we have considered the Tank Allocation Problem in bulk shipping from a tactical perspective where the main objective is to quickly assess feasibility of a given ship route rather than finding an optimal tank allocation. We have developed a heuristic for efficiently solving this problem and computational results show that it can solve 99% of the considered feasible instances within 0.4 seconds and all of them if allowed longer time. We have also modified an optimality based method presented in Hvattum et al. (2009) and thereby improved their results. The heuristic struggles on two instances causing an overall longer average running time than found with this modified optimality based method. Looking instead at the distribution of time usage, we see that when running time is below 10 seconds, our heuristic clearly outperforms the modified optimality based method by consistently solving more instances. This observation motivated the construction of a hybrid method that first runs the heuristic for 0.2 seconds and if no feasible solution is found, then runs the modified optimality based method on the parts of the instance that the heuristic has not solved. Computational results shows that on the considered instances the hybrid method cuts between 90% and 94% of average running times compared to the three other methods and consistently solves more instances than the other methods within any given time limit. In fact, the average running time for the hybrid method is just 0.027 seconds which is fast enough to facilitate the inclusion of the tank allocation aspect into the routing and scheduling phase.

Since the shipping industry operates in a both dynamic and stochastic environment, it is also worth mentioning that, as discussed in Section 8.3.3, our heuristic can be expected to generate solutions that are generally more robust to changes than solutions from the method by Hvattum et al. (2009). This robustness is derived from the fact that the heuristic seeks to allocate each individual cargo in a stable manner whereby the stability of the ship does not rely on cargoes to outbalance each other and hence any cargo can in principle be removed from the route without causing instability for the remaining route.

It should also be mentioned that the heuristic described here is flexible enough to incorporate operational considerations such as ballast tanks and moving cargoes between tanks after allocation (i.e. solving the allocation problem for a single set of cargoes on a leg). Finally, it would be interesting to extend our heuristic to allow flexible cargo sizes since this is often used when shipping liquid products. Likewise, it would be interesting to explore the effect of integrating our hybrid method as a subproblem solution method in a procedure for solving the full routing and scheduling problem with the tank allocation aspect included. Both these extensions are left as promising ideas for further research.

Acknowledgements

The research presented in this paper has been partly funded by The Danish Maritime Fund and we gratefully acknowledge their financial support. This research has also been partially supported by the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under grant agreement n° 246647 and from the New Zealand Government as part of the OptALI project.

Bibliography

D. Barbucha and W. Filipowicz. Segregated storage problem in maritime transportation. In *IFAC Transportation Systems*, pages 557 – 561, 1997.

M.E. Coccola and C.A. Mendez. Logistics management in maritime transporation systems. *Chemical Engineering Transactions*, 32:1291 – 1296, 2013.

F. Cornillier, G. Laporte, F.F. Boctor, and J. Renaud. The petrol station replenishment problem with time windows. *Computers & Operations Research*, 36:919 – 935, 2009.

U. Derigs, J. Gottlieb, J. Kalkoff, M. Piesche, F. Rothlauf, and U. Vogel. Vehicle routing with compartments: applications, modelling and heuristics. *OR Spectrum*, 33:885 – 914, 2011.

K. Fagerholt and M. Christiansen. A combined ship scheduling and allocation problem. *Journal of the Operational Research Society*, 51:834 – 842, 2000.

L.M. Hvattum, K. Fagerholt, and V.A. Armentano. Tank allocation problems in maritime bulk shipping. *Computers & Operations Research*, 36(11):3051–3060, 2009.

A.S. Jetlund and I.A. Karimi. Improving the logistics of multi-compartment chemical tankers. *Computers and Chemical Engineering*, 28:1267 – 1283, 2004.

K. Kobayashi and M. Kubo. Optimization of oil tanker schedules by decomposition, column generation, and time-space network techniques. *Japan Journal of Industrial and Applied Mathematics*, 27(1):161–173, 2010.

G.L. Martin, S.U. Randhawa, and E.D. McDowell. Computerized container-ship load planning: A methodology and evaluation. *Computers & Industrial Engineering*, 14(4):429–440, 1988.

K.-H. Neo, H.-C. Oh, and I.A. Karimi. Routing and cargo allocation planning of a parcel tanker. In *16th European Symposium on Computer Aided Process Engineering and 9th International Symposium on Process Systems Engineering*, pages 1985 – 1990, 2006.

H.-C. Oh and I.A. Karimi. Routing and scheduling of parcel tankers: a novel solution approach. In A. Bruzzone, F. Longo, Y. Merkuriev, G. Mirabello, and M.A. Piera, editors, *The 11th International Workshop on Harbor Maritime Multimodal Logistics Modeling and Simulation*, pages 98 – 103, September 2008.

P. Schaus, J.-C. Regin, R. Van Schaeren, W. Dullaert, and B. Raa. Cardinality reasoning for bin-packing constraint: Application to a tank allocation problem. In M. Milano, editor, *Constraint Programming 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 815 – 822. Springer, 2012.

UNCTAD. Review of maritime transport 2012. `http://unctad.org/en/PublicationsLibrary/rmt2012_en.pdf`, November 2012.

G.A. Vouros, T. Panayiotopoulos, and C.D. Spyropoulos. A framework for developing expert loading system for product carriers. *Expert Systems With Applications*, 10(1):113 – 126, 1996.

I.D Wilson and P.A Roach. Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Research Society*, 51(11):1248–1255, 2000.

X. Wu, H.-C. Oh, I.A. Karimi, M. Goh, and R. de Souza. Tops: Advanced decision support system for port and maritime chemical logistics. *The Asian Journal of Shipping and Logistics*, 27(1): 143 – 156, 2011.

Chapter 9

Tramp Ship Routing and Scheduling with Voyage Separation Requirements

Charlotte Vilhelmsen Richard M. Lusby

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

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

9.1 Introduction

With over 9 billion tons of cargo transported by the international shipping industry every year (UNCTAD, 2013) there is no doubt that the maritime sector plays a vital role in world trade. Maritime transportation therefore constitutes an important research area and in this paper we explore tramp shipping where ships sail much like taxies following the available demand. This is in contrast to liner shipping where ships operate more like busses following a fixed route network and

a published timetable. Tramp operators do however know some of their demand in advance since they can enter into long term contracts and then seek to maximise profit from optional cargoes found in the spot market.

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

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

The remainder of the paper is organized as follows. In Section 9.2 relevant literature is presented while Section 9.3 provides a problem description as well as a non-linear mathematical model for the problem. Section 9.4 describes the decomposition of the problem and the dynamic column generation procedure, i.e. the pricing part of the proposed algorithm. Section 9.5 describes the branching part of the algorithm, namely time window branching and constraint branching. In Section 9.6, we describe the data instances used to evaluate the performance of our algorithm as well as the results obtained with our algorithm on these instances. We also compare these results to results obtained from running the a priori path generation method from Norstad et al. (2013) on these same instances. Finally, concluding remarks and suggestions for future work are discussed in Section 9.7.

9.2 Literature Review

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

Recent work on tramp ship routing and scheduling include Cóccola et al. (2014) who present a novel column generation approach in which the conventional dynamic programming route-generator is replaced by a continuous time MILP slave problem. Their computational results show that their method outperforms both a pure exact optimisation model and a heuristic solution method previously reported in the literature. Castillo-Villar et al. (2014) present a Variable Neighborhood Search based heuristic procedure for solving the tramp ship routing and scheduling problem with discretised time windows. They ignore optional cargoes and therefore seek to minimise cost rather

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

In this paper we extend the basic tramp ship routing and scheduling problem by including voyage separation requirements. Such requirements are also considered in Norstad et al. (2013) in a context similar to ours. Their computational results show that the introduction of voyage separation requirements only leads to a marginal reduction in profit. They present both an arc flow formulation and a path flow formulation solved using a priori path generation. They conclude that both models work well on small problem instances and that the path flow model is also capable of solving real life size instances within an acceptable time. However, for larger instances, or situations with further complexity, neither of the two solution methods are applicable. We investigate the path flow approach further in Section 9.6 where we compare it to our solution method.

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

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

9.3 Problem Description

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

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

port calls for each ship. If capacity is insufficient to perform all mandatory contract voyages, it is possible to charter in spot vessels to perform some of these.

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

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

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

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

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

9.3.1 Mathematical model

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

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

In order to define the problem on a graph, we define an origin and a destination node for each ship $k \in \mathcal{V}$ and denote these $o(k)$ and $d(k)$ respectively. The origin node corresponds to the location of the ship when planning starts while the destination node is artificial and simply corresponds to the geographical location of ship k at the end of the planning horizon. The problem can then be defined on the graph $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \mathcal{N}_C \cup \mathcal{N}_O \cup_{k\in\mathcal{V}} \mathcal{N}_M^k \cup_{k\in\mathcal{V}} \{o(k), d(k)\}$. If a voyage or maintenance, (r, i), can be performed directly before another voyage or maintenance, (q, j), then $\mathcal{A}$ contains the arc $((r, i), (q, j))$. $\mathcal{A}$ also contains the arcs $(o(k), (r, i))$ and $((r, i), d(k))$ for each ship k that can perform voyage i on trade route r. Finally, for each ship $k \in \mathcal{V}$ without maintenance requirements, the set also contains the arc $(o(k), d(k))$ corresponding to an idle ship. For each ship $k \in \mathcal{V}$ we further define the set $\mathcal{A}^k$ as the set of arcs in $\mathcal{A}$ that are traversable by ship k, e.g. with respect to time.

For each node $(r, i) \in \mathcal{N}$ we have a time window $[a_{ri}, b_{ri}]$ describing the earliest and latest time to start service for this voyage or maintenance. For $o(k)$ this window is collapsed into the time 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 $r \in \mathcal{R}$.

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

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

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

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

$$\max \sum_{k\in\mathcal{V}}\sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r}\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q}\left(P^k_{ri} - C^k_{riqj}\right)x^k_{riqj} + \sum_{k\in\mathcal{V}}\sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r} P^k_{ri}x^k_{rid(k)} - \sum_{k\in\mathcal{V}}\sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r} C^k_{o(k)ri}x^k_{o(k)ri} - \sum_{(r,i)\in\mathcal{N}_C} C^S_{ri}y_{ri} \tag{9.1}$$

s.t.

$$\sum_{k\in\mathcal{V}_r}\Big(\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)}\Big) + y_{ri} = 1, \qquad \forall (r,i)\in\mathcal{N}_C, \tag{9.2}$$

$$\sum_{k\in\mathcal{V}_r}\Big(\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)}\Big) \le 1, \qquad \forall (r,i)\in\mathcal{N}_O, \tag{9.3}$$

$$\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)} = 1, \qquad \forall k\in\mathcal{V},\ (r,i)\in\mathcal{N}^k_M, \tag{9.4}$$

$$\sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r} x^k_{o(k)ri} + x^k_{o(k)d(k)} = 1, \qquad \forall k\in\mathcal{V}, \tag{9.5}$$

$$x^k_{o(k)ri} + \sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{qjri} - \sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{riqj} - x^k_{rid(k)} = 0, \qquad \forall k\in\mathcal{V},\ r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{9.6}$$

$$\sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r} x^k_{rid(k)} + x^k_{o(k)d(k)} = 1, \qquad \forall k\in\mathcal{V}, \tag{9.7}$$

$$x^k_{o(k)qj}(t^k_{o(k)} + T^k_{o(k)qj} - t^k_{qj}) \le 0, \qquad \forall k\in\mathcal{V},\ q\in\mathcal{R}^k,\ j\in\mathcal{I}_q, \tag{9.8}$$

$$x^k_{riqj}(t^k_{ri} + T^k_{riqj} - t^k_{qj}) \le 0, \qquad \forall k\in\mathcal{V},\ ((r,i),(q,j))\in\mathcal{A}^k, \tag{9.9}$$

$$a_{ri}\Big(\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)}\Big) \le t^k_{ri} \le b_{ri}\Big(\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q} x^k_{riqj} + x^k_{rid(k)}\Big), \qquad \forall k\in\mathcal{V}, r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{9.10}$$

$$\sum_{k\in\mathcal{V}_r} t^k_{ri} + t^S_{ri}y_{ri} + B_r \le \sum_{k\in\mathcal{V}_r} t^k_{r,i+1} + t^S_{r,i+1}y_{r,i+1}, \qquad \forall (r,i)\in\mathcal{N}_C, i\in\mathcal{I}_r\setminus\{n_r\}, \tag{9.11}$$

$$a_{ri} \le t^S_{ri} \le b_{ri}, \qquad \forall (r,i)\in\mathcal{N}_C, \tag{9.12}$$

$$t^k_{o(k)} \ge a_{o(k)}, \qquad \forall k\in\mathcal{V}, \tag{9.13}$$

$$x^k_{o(k)ri}\in\{0,1\}, \qquad \forall k\in\mathcal{V},\ r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{9.14}$$

$$x^k_{o(k)d(k)}\in\{0,1\}, \qquad \forall k\in\mathcal{V}:\mathcal{N}^k_M=\emptyset, \tag{9.15}$$

$$x^k_{riqj}\in\{0,1\}, \qquad \forall k\in\mathcal{V},\ ((r,i),(q,j))\in\mathcal{A}^k, \tag{9.16}$$

$$x^k_{rid(k)}\in\{0,1\}, \qquad \forall k\in\mathcal{V},\ r\in\mathcal{R}^k,\ i\in\mathcal{I}_r, \tag{9.17}$$

$$y_{ri}\in\{0,1\}, \qquad \forall (r,i)\in\mathcal{N}_C. \tag{9.18}$$

The objective function (9.1) 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 (9.2) and (9.3) 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 (9.4) ensure that these requirements are adhered to. Constraints (9.5) and (9.7) together with the flow conservation constraints in (9.6) ensure that each ship is assigned a schedule starting at the origin node and ending at the destination node. Constraints (9.8) ensure that if the schedule for ship k visits node $o(k)$ directly before node (q,j), the service at node (q,j) cannot begin before service time at node $o(k)$ plus travel time from node $o(k)$ to node (q,j) with ship k. Since waiting time is allowed, the constraints have an inequality sign. Constraints (9.9) 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 (9.10), 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 (9.11) 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 (9.12).

Constraints (9.13) ensure that no ship can start its schedule before it is available for service. The flow variables are restricted to be binary in (9.14)-(9.17) while constraints (9.18) impose similar restrictions on the spot vessel decision variables. Note that due to constraints (9.2) and the binary restrictions on the flow variables, we do not actually need the binary restrictions on the spot vessel variables. However, we include them for completeness sake and also to exploit their binary nature in the branch-and-bound scheme later.

9.4 Decomposition

The nonlinear mixed integer programming model (9.1)-(9.18) could in theory be solved by commercial optimization software for linear problems after linearising constraints (9.8), (9.9), and (9.11). However, as pointed out in Norstad et al. (2013), where they use a similar arc flow model, most real life problem instances will be too large to achieve solutions in a reasonable amount of time. This section therefore describes a solution method tailored for the TSRSPVSR.

In the mathematical programming model (9.1)-(9.18), constraints (9.4)-(9.10) and (9.13)-(9.17) 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 (9.2), (9.3) and (9.11) 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 (9.12) and (9.18) remain in the master problem in which feasible ship schedules constitute the columns. This way the original problem is transformed into a master problem with a reduced number of constraints but with a potentially very large number of columns.

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

9.4.1 Master Problem

The common constraints (9.2), (9.3) and (9.11) in combination with the spot vessel constraints (9.12) and (9.18) and the objective function (9.1) constitute the master problem. They must, however, be expressed by new path flow variables corresponding to feasible ship schedules and constraints must be added to ensure that each ship is assigned exactly one schedule. We let $\mathcal{S}^k$ denote the set of all feasible schedules for ship k.

Note that rather than selecting exactly one distinct schedule for each ship, we can also allow convex combinations of different schedules for each ship, as long as the chosen schedules correspond to the same geographical route. We therefore denote the set of geographical routes for ship k by $\mathcal{G}^k$. Furthermore, we expand notation on the schedule sets so that now $\mathcal{S}_g^k$ 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_s^k for $k \in \mathcal{V}$, $g \in \mathcal{G}^k$, $s \in \mathcal{S}_g^k$, and define a binary schedule variable λ_s^k that is equal to 1 if ship k is chosen to sail schedule s, and 0 otherwise. The profit p_s^k is calculated based on information from the underlying schedule, which holds all necessary information, i.e. the ship it is constructed for, the voyages conducted, and the timing of port calls during the schedule. We reuse the definition of y_{ri} from the arc flow formulation and let A_{ris}^k 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_{ris}^k. 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:

$$\max \sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}_g^k} p_s^k \lambda_s^k - \sum_{(r,i)\in\mathcal{N}_C} C_{ri}^S y_{ri} \tag{9.19}$$

s.t.

$$\sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}_g^k} A_{ris}^k \lambda_s^k + y_{ri} = 1, \qquad \forall (r,i) \in \mathcal{N}_C, \tag{9.20}$$

$$\sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}_g^k} A_{ris}^k \lambda_s^k \leq 1, \qquad \forall (r,i) \in \mathcal{N}_O, \tag{9.21}$$

$$\sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}_g^k} \lambda_s^k = 1, \qquad \forall k \in \mathcal{V}, \tag{9.22}$$

$$\sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}_g^k} T_{ris}^k \lambda_s^k + t_{ri}^S y_{ri} + B_r \leq \sum_{k\in\mathcal{V}} \sum_{g\in\mathcal{G}^k} \sum_{s\in\mathcal{S}_g^k} T_{r,i+1,s}^k \lambda_s^k + t_{r,i+1}^S y_{r,i+1}, \quad \forall (r,i) \in \mathcal{N}_C, i \in \mathcal{I}_r \setminus \{n_r\}, \tag{9.23}$$

$$a_{ri} \leq t_{ri}^S \leq b_{ri}, \qquad \forall (r,i) \in \mathcal{N}_C, \tag{9.24}$$

$$\sum_{s\in\mathcal{S}_g^k} \lambda_s^k \in \{0,1\}, \qquad \forall k \in \mathcal{V},\, g \in \mathcal{G}^k, \tag{9.25}$$

$$y_{ri} \in \{0,1\}, \qquad \forall (r,i) \in \mathcal{N}_C. \tag{9.26}$$

To solve the problem in an LP based branch-and-price framework, we first relax the binary constraints on the decision variables. The VSR constraints (9.23) will complicate the subproblems as the dual variables of these constraints will create linear node costs in the subproblems. Furthermore, as T_{ris}^k is a non-binary parameter, the presence of the VSR constraints in the master problem will most likely lead to more fractional solutions as it compromises the strong integer properties of the constraint matrix (we return to this matter in Section 9.5.2). Therefore, we also relax the VSR constraints (9.23) 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 (9.24) and so we also relax these.

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

schedules for each ship, and iteratively generate new columns that have the potential to improve the current solution. The entire branch-and-price process therefore begins with the solution of the *restricted master problem* (RMP) which is the linear relaxation of the problem (9.19)-(9.22) but with only a subset of the columns included. The dual variables from this solution are then used in the subproblems, also called pricing problems, to generate new promising columns that are added to the RMP. This iterative process continues until no further columns can improve the current master problem solution. If the current solution to the RMP is infeasible with respect to either or both the relaxed integrality and VSR constraints, we branch and continue this entire process by pricing out new columns at each node of the search tree until a feasible, optimal solution is obtained, or a specified time limit elapses.

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

9.4.2 Subproblems - Pricing out new schedules

Constraints (9.4)-(9.10) and (9.13)-(9.17) split into one independent subproblem for each ship. Since these are all essentially the same problem, we simply consider the generic subproblem for ship k and refer to 'the subproblem'. Note that there is interdependence between the subproblems due to the common constraints. The ship routing constraints in the subproblem ensure that any solution is a feasible schedule for ship k and the objective must ensure that only schedules with the potential to improve the current solution of the RMP are generated. This means finding schedules with positive reduced costs in the current solution of the RMP.

Let π_{ri} be the dual variables for constraints (9.20) and (9.21) where the variables corresponding to (9.20) are free of sign while the variables corresponding to (9.21) must be nonnegative. Next, define $\pi_{ri} = 0$ for all $(r, i) \in \mathcal{N}_M^k$ and let ω_k be the dual for constraint (9.22) which is also free of sign. Since we consider the generic subproblem we can drop the superscript k on the variables and the subproblem is then given by:

$$\begin{aligned}\max \sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r}\sum_{q\in\mathcal{R}^k}\sum_{j\in\mathcal{I}_q}\big(P^k_{ri} - C^k_{riqj} - \pi_{ri}\big)x_{riqj} \\ + \sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r}\big(P^k_{ri} - \pi_{ri}\big)x_{rid(k)} \\ - \sum_{r\in\mathcal{R}^k}\sum_{i\in\mathcal{I}_r} C^k_{o(k)ri}x_{o(k)ri} - \omega_k \qquad (9.27)\end{aligned}$$

s.t.

$$(9.4) - (9.10) \text{ and } (9.13) - (9.17). \qquad (9.28)$$

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

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

Subproblem networks

During construction of the subproblem networks, standard preprocessing techniques are applied to tighten the time windows. The VSR constraints are especially useful in this process since two consecutive voyages on a contract trade, r, must be separated in time by at least B_r and so, their individual time windows must reflect this. We describe this time window reduction technique in further details in Section 9.5.1.

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

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

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

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

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

As already mentioned, we expect few if any time feasible cycles in our data instances. To detect the presence of potential cycles in a given subproblem network, a topological sort is performed on the node set. If a cycle is detected, the involved nodes are split into several duplicate ones, each with a smaller time window, until the cycle no longer exists; this process creates an acyclic network. It should be noted that with such a network we can generate schedules that visit multiple of these duplicate nodes, whereby the cycle in effect still exists. For maintenance nodes, there is however no profit. Therefore, we will never want to visit such nodes in the first place, and the maintenance resource ensures that we will visit exactly one of these. Note that if such cyclic schedules are added to the master problem, we can have $A_{ris} > 1$ in the formulation of the master problem (9.19)-(9.26).

Dynamic Programming Algorithm

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

$$\hat{C}_{o(k),(r,i)} = C^k_{o(k)ri} + \omega_k \qquad \forall\,(o(k),(r,i)) \in \mathcal{A}^k, \tag{9.29}$$

$$\hat{C}_{(r,i),(q,j)} = C^k_{riqj} - P^k_{ri} + \pi_{ri} \qquad \forall\,((r,i),(q,j)) \in \mathcal{A}^k, \tag{9.30}$$

$$\hat{C}_{(r,i),d(k)} = -P^k_{ri} + \pi_{ri} \qquad \forall\,((r,i),d(k)) \in \mathcal{A}^k. \tag{9.31}$$

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

Two partial schedules generated at the same node can be compared by defining a partial order relation between the respective labels. This partial order allows us to determine if one label dominates another that can, hence, be discarded. This dominance concept ensures that only the best schedules, i.e. Pareto optimal, are kept during the iterative process of the algorithm as only they can contribute to the optimal schedule. It is important to note that here "optimal" refers to the LP relaxed master problem, *without* constraints (9.23). The success of the algorithm relies on an efficient domination procedure for the labels to eliminate non-useful partial schedules. In our case we note that a schedule s_n ending at node n dominates another schedule s'_n also ending at node n if and only if $\mathcal{L}(s_n) \neq \mathcal{L}(s'_n)$, $\bar{C}(s_n) \leq \bar{C}(s'_n)$, $T(s_n) \leq T(s'_n)$ (since there is no cost for waiting) and $M(s_n) \geq M(s'_n)$.

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

The dynamic programming algorithm we implement is hence a standard label setting algorithm, which begins at $o(k)$ with an initial label. Nodes are considered in topological order, and processed in turn. In processing a node, all non-dominated labels for the current node are extended, using the resource extension functions defined above and consider the node's set of outgoing arcs. When the algorithm terminates, several resource feasible and Pareto optimal schedules might exist differing in both reduced cost and time. In Section 9.4.3 we discuss what to do with these schedules. See Algorithm 5 for a general overview of our label setting algorithm.

9.4.3 Pricing Strategy

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

Algorithm 5: Label Setting Algorithm

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

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

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

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

9.5 Branching

If the optimal solution to the restricted master problem is both integer (λ does not have to be integer as long as all positive λ's for each ship correspond to the same geographical route) and fulfills all the VSR constraints (9.23) and time window restrictions for spot vessels (9.24), the solution is also optimal for the full master problem (9.19)-(9.26) and thereby also for the original problem (9.1)-(9.18). 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 (9.19)-(9.26).

9.5.1 Time Window Branching

The overall idea in time window branching is to split a given time window into two smaller time windows that each correspond to a new problem, i.e. to a new branch in the branch-and-bound tree. The trick is to select the time window and the split time in such a way that the current solution becomes infeasible in each of the two new problems, i.e. in a way that makes at least one chosen schedule infeasible in each branch. Gélinas et al. (1995) note that their method works best on problems with small time windows and few cycles in the linear relaxation solution. As already mentioned, we expect few if any cycles in our data instances. However, time windows are relatively wide and so, it remains to be investigated if time window branching can work well for

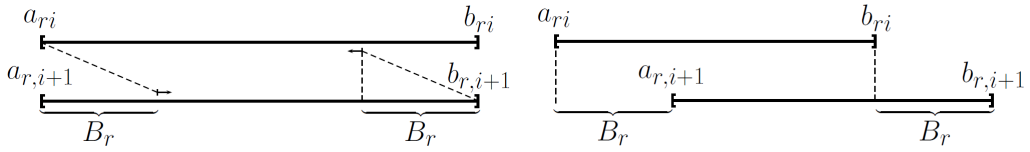
our problem. The method described by Gélinas et al. (1995) was developed to restore integrality and does not factor in VSR constraints. Therefore, we extend their method to incorporate such constraints just as e.g. Dohn et al. (2011) and Rasmussen et al. (2012) have done it to accommodate temporal dependencies for, respectively, vehicle routing and home care crew scheduling. Furthermore, we extend their methods to also account for the spot vessel time window constraints (9.24) and use a slightly modified approach that will improve efficiency of the branching scheme. Note that without the VSR constraints (9.23), 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.

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 9.1 therefore states the possible time window reductions for two consecutive nodes (r, i) and $(r, i+1)$ on trade r. The reduction process is illustrated in Figure 9.1.

Table 9.1: Time window reduction rule

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



(a) Before reduction (b) After reduction

Figure 9.1: Time window reduction process for two consecutive nodes on a trade

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

Candidate time windows

Any node with a time window that can be split in a way that renders at least one currently chosen schedule infeasible in each of the two new branches, is a candidate for branching. If the current master problem solution is fractional, then to fulfill constraints (9.20) or (9.21) there must be a node i (omitting the trade index r for now) that is visited by more than one schedule, one of them possibly a spot vessel schedule, or several times in a cycle by the same schedule. Hence, we must split the time window at this node so that there is only a single visit to the node. For each fractional

schedule that visits node i, it might be possible to change the start time at node i slightly without rendering the schedule infeasible. For each visit to node i, we determine a *feasibility interval* defined by the earliest and latest possible start time at this node that will allow the corresponding schedule to remain feasible. For spot vessel schedules the feasibility interval simply corresponds to the (possibly reduced) time window at the node. The label setting algorithm used to generate schedules for fleet vessels, schedules each node visit as early as possible. This means that the feasibility interval at each node for regular schedules will simply correspond to allowing the ship to wait at long as possible. Assume now that node i is visited twice by two different schedules or twice by the same schedule. It does not matter if the two visits correspond to the same ship or to different ships, and so we can omit the k index here. Therefore, we abuse notation slightly by letting T_i^1 and T_i^2 denote the visit time for these two visits respectively. Now let $[T_i^1, u_1]$ and $[T_i^2, u_2]$ be the corresponding feasibility intervals and let $\epsilon > 0$ be a very small tolerance. If these two intervals are disjoint as shown in Figure 9.2, we can choose a split time for node i in $(u_1, T_i^2]$, say t_s, and create one branch where the time window for node i is $[a_{ri}, t_s - \epsilon]$ and the second visit is infeasible, and one branch where the time window is $[t_s, b_{ri}]$ where the first visit is infeasible. We can generalise this to state that any node for which two visits to the node have disjoint feasibility intervals, is a candidate for branching. A formal description and a proof for this, can be found in Gélinas et al. (1995). Note that split times within one of the feasibility intervals would also render one visit infeasible in each branch. However, during the next iteration of schedule generation, the visit, for which we selected a split time within its feasibility interval, could be regenerated though only a little later in time. This means that in one branch we will actually regenerate a fractional solution similar to the one from the parent node. Hence, it will be ineffective to use time window branching when the feasibility intervals are not disjoint. Note that in both Dohn et al. (2011) and Rasmussen et al. (2012) feasibility intervals are ignored whereby their branching scheme potentially becomes ineffective, both due to regeneration of fractionality but also because this approach causes them to consider more nodes as candidates for branching than if they had included the feasibility intervals. Since each candidate node must be further investigated to determine both the best split time for each node and in turn select the best node, this makes their approach more time consuming.

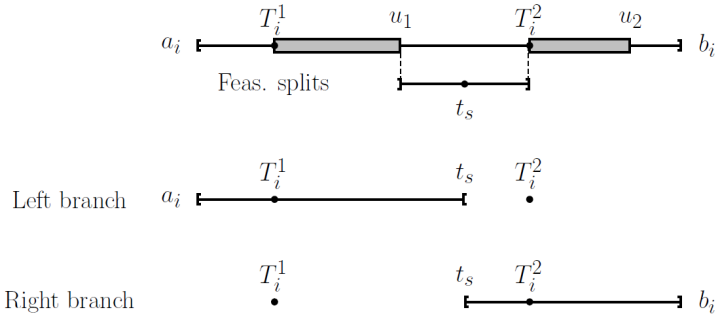


Figure 9.2: Time window branching due to fractionality

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

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

Extending this concept to include VSR constraints and spot vessel time windows, further candidate time windows arise. Assume now that the VSR constraint for two consecutive voyages, i and $i+1$, on trade r is violated and that $y_{ri} = y_{r,i+1} = 0$, i.e. no spot vessels are involved. If the current solution is integral, there is only one visit to each of these two nodes and these two visits per assumption violate the VSR constraint. If the current solution is fractional, there can

be multiple visits to each of nodes (r, i) and $(r, i+1)$. For the VSR constraint to be violated, there must however be at least one pair of visits that on their own violate the VSR constraint. Whether the solution is integral or fractional, this means that there must exist positive $\lambda_{s_1}^{k_1}$ and $\lambda_{s_2}^{k_2}$ in the current RMP solution such that $T_{ris_1}^{k_1} + B_r > T_{r,i+1,s_2}^{k_2}$. Note that it is possible that $s_1 = s_2$ and $k_1 = k_2$. For the branching we generally do not need to know the specific schedule or ship index but simply that two visits for consecutive voyages on the same trade are violating the VSR. Therefore, we continue to abuse notation by letting T_i and T_{i+1} denote the visit times at these two nodes. To restore feasibility of the VSR constraint, we can force the start time of node (r, i) to be scheduled earlier, namely no later than $T_{i+1} - B_r$, we can postpone the start time of node $(r, i+1)$ so that it occurs at the earliest at $T_i + B_r$ or we can use a combination of these two time window alterations. In essence, the above corresponds to splitting the time window for node (r, i) and then reducing the time window of node $(r, i+1)$ accordingly so that the minimum time spread, B_r, is adhered to. Therefore, we use a split time, t_s, in the interval $[T_{i+1} - B_r + \epsilon, T_i]$ and create a left branch where the time window of node (r, i) is restricted to $[a_{ri}, t_s - \epsilon]$ whereby the schedule visiting node i becomes infeasible, and a right branch where the time window for node i is restricted to $[t_s, b_{ri}]$. In the left branch the time window for node $(r, i+1)$ remains unchanged while in the right branch we use the reduction rule from Section 9.5.1 to reduce it to $[t_s + B_r, b_{ri+1}]$ whereby the schedule visiting node $i+1$ becomes infeasible. Figure 9.3 demonstrates this process. Note that the process can be reversed to similarly enable a split of the time window at node (r, i) if instead the VSR violation stems from the node pair $(r, i-1)$ and (r, i).

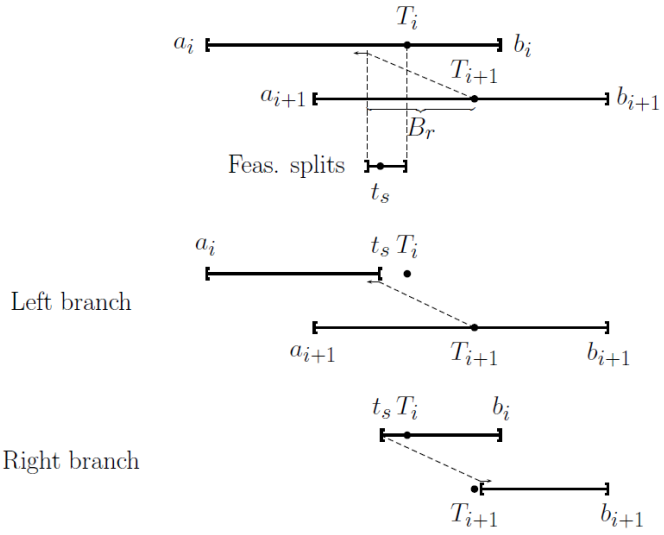


Figure 9.3: Time window branching due to VSR violation

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

1. Branching on non-violated VSR constraints could lead to an ineffective branching scheme

2. Also considering nodes involved in non-violated VSR constraints to be branching candidates, could give rise to excessively long candidate lists. The selection of the best node for branching involves further investigation of each of the node candidates to determine the best split time as well as the flow elimination from this best split time. Therefore, having a long candidate list will be time consuming.

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

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

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i+1,s}\lambda^k_s, \tag{9.32}$$

$$a_{ri} \leq t^S_{ri} \leq b_{ri}. \tag{9.33}$$

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 $((r,i-1),(r,i))$:

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-1,s}\lambda^k_s + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri}, \tag{9.34}$$

$$a_{ri} \leq t^S_{ri} \leq b_{ri}. \tag{9.35}$$

If we insert $t^S_{ri} = a_{ri}$ in (9.34), we might find that the constraint is violated and thereby conclude that the VSR constraint for voyage pair $((r,i-1),(r,i))$ is violated even though the constraint need not be if we just select a different value for t^S_{ri}. Therefore, these two VSR constraints must be considered simultaneously. If $y_{r,i-1}$ is also positive, we must also include the VSR constraint for voyage pair $((r,i-2),(r,i-1))$ along with the spot vessel time window for $t^S_{r,i-1}$ and so it continues until we reach a voyage $(r,i-m)$ for which $y_{r,i-m} = 0$, or until we reach the first voyage on this particular trade. All of these VSR constraints and their corresponding spot vessel time windows form an LP for this particular *VSR group*:

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m,s}\lambda^k_s + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m+1,s}\lambda^k_s + t^S_{r,i-m+1}y_{r,i-m+1}, \tag{9.36}$$

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m+1,s}\lambda^k_s + t^S_{r,i-m+1}y_{r,i-m+1} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-m+2,s}\lambda^k_s + t^S_{r,i-m+2}y_{r,i-m+2}, \tag{9.37}$$

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

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i-1,s}\lambda^k_s + t^S_{r,i-1}y_{r,i-1} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri}, \tag{9.38}$$

$$\sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{ris}\lambda^k_s + t^S_{ri}y_{ri} + B_r \;\leq\; \sum_{k\in\mathcal{V}}\sum_{s\in\mathcal{S}^k} T^k_{r,i+1,s}\lambda^k_s, \tag{9.39}$$

$$a_{r,i-m+1} \leq t^S_{r,i-m+1} \leq b_{r,i-m+1}, \tag{9.40}$$

$$a_{r,i-m+2} \leq t^S_{r,i-m+2} \leq b_{r,i-m+2}, \tag{9.41}$$

$$\vdots \quad \vdots \quad \vdots \tag{9.42}$$

$$a_{r,i-1} \leq t^S_{r,i-1} \leq b_{r,i-1}, \tag{9.43}$$

$$a_{ri} \leq t^S_{ri} \leq b_{ri}. \tag{9.44}$$

If a feasible solution exists to the LP (9.36)-(9.44), the VSR group is not violated. If on the other hand no solution exists, at least one of the following statements must be true:

$$\exists g \in \{0,1,\ldots,m\}, s_1, s_2, k_1, k_2 : \lambda_{s_1}^{k_1} > 0, \lambda_{s_2}^{k_2} > 0, T_{r,i-g,s_1}^{k_1} + B_r > T_{r,i-g+1,s_2}^{k_2}, \quad (9.45)$$

$$\exists g \in \{1,2,\ldots,m\}, s_1, s_2, k_1, k_2 : \lambda_{s_1}^{k_1} > 0, \lambda_{s_2}^{k_2} > 0, T_{r,i-g,s_1}^{k_1} + 2B_r > T_{r,i-g+2,s_2}^{k_2}, \quad (9.46)$$

$$\exists g \in \{2,3,\ldots,m\}, s_1, s_2, k_1, k_2 : \lambda_{s_1}^{k_1} > 0, \lambda_{s_2}^{k_2} > 0, T_{r,i-g,s_1}^{k_1} + 3B_r > T_{r,i-g+3,s_2}^{k_2}, \quad (9.47)$$

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

$$\exists g \in \{m-1,m\}, s_1, s_2, k_1, k_2 : \lambda_{s_1}^{k_1} > 0, \lambda_{s_2}^{k_2} > 0, T_{r,i-g,s_1}^{k_1} + mB_r > T_{r,i-g+m,s_2}^{k_2}, \quad (9.48)$$

$$\exists s_1, s_2, k_1, k_2 : \lambda_{s_1}^{k_1} > 0, \lambda_{s_2}^{k_2} > 0, T_{r,i-m,s_1}^{k_1} + (m+1)B_r > T_{r,i+1,s_2}^{k_2}. \quad (9.49)$$

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

$$T_{r,i-2,s_1}^{k_1} + B_r > T_{r,i-1,s_2}^{k_2} \quad (9.50)$$

$$T_{r,i-1,s_1}^{k_1} + B_r > T_{ris_2}^{k_2} \quad (9.51)$$

$$T_{ris_1}^{k_1} + B_r > T_{r,i+1,s_2}^{k_2} \quad (9.52)$$

$$T_{r,i-2,s_1}^{k_1} + 2B_r > T_{ris_2}^{k_2} \quad (9.53)$$

$$T_{r,i-1,s_1}^{k_1} + 2B_r > T_{r,i+1,s_2}^{k_2} \quad (9.54)$$

$$T_{r,i-2,s_1}^{k_1} + 3B_r > T_{r,i+1,s_2}^{k_2} \quad (9.55)$$

Constraint (9.52) corresponds to the illustration in Figure 9.3 and this figure must now be extended to include the time windows and visit times corresponding to constraints (9.51) and (9.53). The remaining violations will be handled when branching on nodes $(r,i-2)$, $(r,i-1)$ and $(r,i+1)$.

Spot vessels act as a shipping specific version of uncovered tasks, and these are not considered in Dohn et al. (2011), and in Rasmussen et al. (2012) they do not require uncovered tasks to be within the time windows. With our inclusion of spot vessels and time window restrictions it would be extremely time consuming to use the approach of Dohn et al. (2011) and Rasmussen et al. (2012) to run through all nodes and search for individual visit times that violate the constraints corresponding to the VSR group, i.e. constraints (9.45)-(9.49). Instead we run through each VSR group and only for the violated ones, we search for visit times that fulfill constraints (9.45)-(9.49), starting from (9.45). If we find one or several pairs of visit times that fulfill constraints (9.45), we have one or several pairs of candidate nodes for branching. In that case we do not check constraints (9.46)-(9.49) since these can be implicitly handled by branching to fulfill constraints (9.45) and using the time window reduction rule. However, if we do not find a pair of visit times that fulfill constraints (9.45), then we move on to check constraints (9.46) and so the process continues until we find branching candidates.

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

Selecting the best split time within a time window

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

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

for a given node, we consider the feasibility intervals of all visits to the node, again excluding visits from spot vessels. We illustrate the process in Figure 9.4 for a situation where node i is visited by four schedules included in the current solution with fractional values. We denote the four visit times at node i, respectively, T_i^1, T_i^2, T_i^3 and T_i^4. In Figure 9.4 the grey boxes represent the feasibility intervals for each of the four fractional visits to node i.

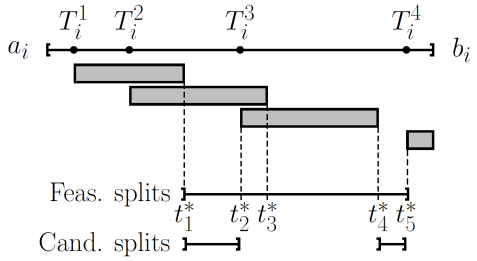


Figure 9.4: Choosing a split time to restore integrality

In order to render the current solution infeasible in both new branches, at least one of the four schedules must become infeasible in each of these branches. Therefore, the split time must be chosen somewhere in the interval between two disjoint feasibility intervals; hence, as depicted by the line marked 'Feas. splits' in Figure 9.4. Note that split times in this interval lie outside the feasibility interval corresponding to visit time T_i^1. Thereby, we can never regenerate a similar visit in the right branch, where visit time T_i^1 becomes infeasible. Comparing the intervals $(t_2^*, t_3^*]$, $(t_3^*, t_4^*]$ and $(t_4^*, t_5^*]$ we see that split times within these three intervals have the same affect with respect to immediate flow elimination. However, if we select a split time in $(t_2^*, t_3^*]$ we risk regenerating the schedules corresponding to both T_i^2 and T_i^3 by redistributing waiting time. However, if instead we select a split time in $(t_3^*, t_4^*]$, we cannot regenerate the schedule corresponding to T_i^2 and selecting a split time in $(t_4^*, t_5^*]$ we cannot regenerate either of the schedules corresponding to T_i^2 and T_i^3. In order to avoid regenerating schedules that lead to fractionality, we will therefore never consider selecting a split time within $(t_2^*, t_4^*]$. More generally, the open ended candidate intervals for split time should not include starting or end points of feasibility intervals and we should seek split times on the boundaries of the feasibility intervals. Thereby, we generally seek split times that lie outside the feasibility intervals. Hence, we can reduce the set of candidate split times to the two smaller intervals shown in Figure 9.4 as 'Cand. splits'. Within each of these two intervals the amount of flow eliminated is independent of the chosen split time. However, as already discussed, the label setting algorithm used to generate the schedules, ensures that each visit is scheduled as early as possible. Thereby, no visit can be regenerated at an earlier point in time but it can however be postponed. Selecting the split time as late as possible within a given candidate interval therefore generally minimises the risk that an eliminated visit will be regenerated. In Figure 9.4 this means that in the interval $(t_1^*, t_2^*]$ we select t_2^* as the split time while in $(t_4^*, t_5^*]$ we select t_5^*. Note that t_2^* and t_5^* both correspond to visit times. We generalise this to say that each currently selected visit time, T_i, after the feasibility interval of the earliest selected visit at the node, is a candidate split time. By using this approach, we only need to calculate the feasibility interval of the first visit to the node.

Now we extend to include VSR violations and we assume that nodes $(r, i-1)$, (r, i) and $(r, i+1)$ together form a VSR group. Figure 9.5 therefore extends the example from Figure 9.4 to include VSR violations for node pair $(r, i-1)$ and (r, i) as well as for node pair (r, i) and $(r, i+1)$. Here T_{i-1}^1 and T_{i+1}^1 denote, respectively, the visit time of a visit to node $i-1$ and $i+1$ for two schedules included in the current solution. Note that for VSR violations we do not care about feasibility intervals since it is the exact visit time at the node that impacts the VSR constraint. Therefore, the feasibility intervals are not included in Figure 9.5. There are four VSR violations and each of these leads to an interval of feasible split times. These are marked 'Feasible split intervals' in the bottom of the figure. The start and end points of these four intervals are marked as t_6^* to t_{11}^* and together they define five intervals, i.e. $(t_6^*, t_7^*]$ to $(t_{10}^*, t_{11}^*]$, which each have a distinct elimination of flow. Using the same reasoning as above, within each of these five intervals we prefer to select

the latest time. This means that in Figure 9.5, t_7^* to t_{11}^* are all candidates for split times. We note that t_7^*, t_{10}^* and t_{11}^* all correspond to visit times at the node while t_8^* and t_9^* correspond to, respectively, $T_{i+1}^1 - B_r$ and $T_{i-1}^1 + B_r$. We can generalise this to say that, limiting the search to schedules included in the current master problem solution with a positive value, the start time of every visit to the node except the first is a candidate split time. Furthermore, for any visit at node $i-1$ for which the visit time T_{i-1} causes a violation of the VSR for nodes $i-1$ and i, $T_{i-1}^1 + B_r$ is also a candidate split time. Similarly, any visit at node $i+1$ for which the visit time T_{i+1} causes a violation of the VSR for nodes i and $i+1$, $T_{i+1} - B_r$ is a candidate split time. Furthermore, for various integer values of m, depending on the size of the VSR group that node i belongs to, $T_{i\pm m} \pm mB_r$ can also be candidate split times. Note that this extends the candidate split times derived from fractionality since all start times at the node except the first one is now a candidate split time.

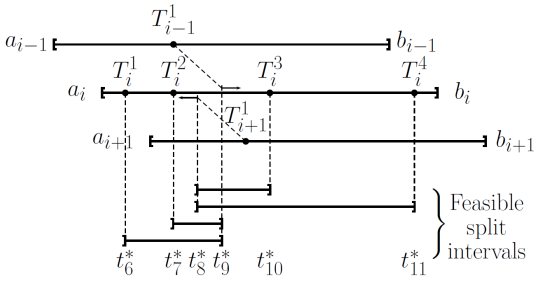


Figure 9.5: Choosing a split time to restore VSR

Assume now that the schedules corresponding to T_i^1, T_i^2, T_i^3 and T_i^4 are in the current solution with values 0.1, 0.1, 0.4 and 0.4, respectively. Furthermore, assume that the two visits at nodes $i-1$ and $i+1$ are the only visits to these nodes, i.e. that the schedules corresponding to these visits are both in the solution with a value of 1. Table 9.2 then shows the infeasible visits and the corresponding eliminated flow in, respectively, the left and right branch when choosing one of the candidate split times derived from Figure 9.5 (which also covers the candidate split times from Figure 9.4).

Note that since the branching applied to restore feasibility with respect to VSRs only factors in the exact visit times and ignores the feasibility intervals, this type of branching allows us to regenerate similar schedules with visit times just slightly postponed. This is sufficient for the VSR constraints, however it might not be sufficient to rule out regeneration of fractionality. As an example, consider the split time candidate $t_7^* = T_i^2$ where the right branch will exclude visit time T_i^1. According to Figure 9.4, T_i^2 is within the feasibility interval of the schedule, say s_1, corresponding to visit time T_i^1. Therefore, we can easily generate a new schedule similar to s_1 with the visit at node i postponed slightly. Thereby, the exact same fractionality will occur again. Therefore, when branching on a node where there is no VSR violations, we reinclude the feasibility interval aspect to more effectively eliminate fractionality.

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 9.2 in column 'Minimum' for each candidate, and we see that the best worst case flow elimination is achieved for t_9^*.

Note that Table 9.2 only includes immediate flow elimination and not the implicit flow 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 very 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.

Table 9.2: Flow elimination from candidate split times

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

Selecting the best time window for branching

Now that we know how to find candidate time windows for branching and also how to actually split the chosen time window, we are ready to choose which time window to branch on. Again, aiming at eliminating as much flow as possible while maintaining a well balanced search tree, we select the time window that has the best worse case flow elimination. I.e. the approach from Table 9.2 for selecting the best split time for a given time window, is also used to select the best time window to split.

9.5.2 Constraint Branching

For now, ignore the possible existence of cyclic master problem schedules, i.e. columns with $A_{ris} > 1$ in (9.19)-(9.26). If slack variables, y_{ri}, are inserted into constraints (9.21), the RMP is modelled as a set partitioning problem with generalised upper bound constraints (9.22). Due to the generalised upper bound constraints (9.22), the submatrix for each ship is perfect. Thereby, fractional solutions can only appear across submatrices for different ships and never within one of the individual ship submatrices. This means that the LP solution can only be fractional if two or more ships are competing for the same voyage. Note the word 'voyage' since ships can never compete for the individual maintenance requirements. We refer the reader to Padberg (1973) and Conforti et al. (2001) for a discussion on perfect matrices and their properties. This strong integer property of the RMP constraint matrix means that the upper bounds in the branch-and-bound algorithm are very tight and that we can expect to reach integral solutions after only a few iterations of branching. We exploit this underlying structure of the constraint matrix in the branching scheme to apply so called constraint branching, see Ryan and Foster (1981). Note that this strong integer property also means that we can refrain from checking for 'equal geographical routes' for each ship.

Candidate voyage-ship pairs

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

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

If the current solution is fractional, there must exist a voyage (r, i) and a ship k for which $0 < S_{ri}^k < 1$. The branching strategy is then to construct a left branch where ship k is forced to

perform voyage (r, i) and a right branch where ship k is not allowed to perform voyage (r, i). In the left branch this means that $A_{ris} = 1$ for all $s \in \mathcal{S}^k$. Thereby, we can remove all schedules for k that do not include voyage (r, i) and all columns for other ships that do include voyage (r, i). This also means removing the spot vessel schedule for voyage (r, i) or equivalently setting $y_{ri} = 0$. Furthermore, we remove voyage (r, i) in all subproblems networks not corresponding to ship k. We rely on the dual variables to eventually enforce the construction of schedules for ship k that include voyage (r, i). In the right branch, we instead have $A_{ris} = 0$ for all $s \in \mathcal{S}^k$. This is the opposite process of the left branch, and so we instead remove all schedules for k that *do* include voyage (r, i). Furthermore, we remove voyage (r, i) in the subproblem network corresponding to ship k while we cannot make any changes in the networks corresponding to other ships.

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

Selecting the voyage-ship pair for branching

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

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

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

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

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

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. Thereby, fractionality can occur without competition between different ships, i.e. we can have a fractional solution while $\mathcal{C} = \emptyset$ so that no branching candidates exist. Hence, the above constraint branching scheme cannot completely eliminate fractionality derived from cyclic schedules and we must instead rely on time window branching to eliminate fractionality derived from such schedules.

9.5.3 Branching Strategy

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

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

The first strategy initially uses constraint branching to eliminate fractionality and then turns to time window branching to handle violations of VSR constraints. Note that in this case, assuming that no cyclic schedules are used, the time window branching scheme will be simplified since only one visit to each node exists and we do not have to consider feasibility intervals. The second strategy uses time window branching to handle both VSR violations and fractionality and only turns to constraint branching if we run out of time windows to branch on. The third strategy starts by handling all violations of VSR constraints while ignoring fractionality, i.e. disregarding feasibility intervals and in general nodes that are not involved in VSR violations. Afterwards, all fractionality is eliminated through constraint branching. Note though that the VSR related time window branching will simultaneously help to eliminate fractionality and that time window branching must be used if fractionality occurs due to cyclic schedules.

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

9.6 Computational Study

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

9.6.1 Data instances

To properly evaluate the performance of the devised algorithm, we test it on 16 problem instances of varying complexity and size. These instances have been generated by the test instance generator described in Norstad et al. (2013) which is based on industry data. Table 9.3 presents the main characteristics of the 16 data instances. The column labels are almost self explanatory but for completeness sake we note that from left to right they give the instance number, the number of ships, the number of trades, the number of voyages where the number in parenthesis gives the number of spot voyages and finally, the length of the planning horizon in days. Note that this horizon is defined as the length of the period that contains the earliest allowed starting time for each voyage. Thereby, planning will continue well beyond this horizon since voyages can be performed later than the earliest allowed time and must also be completed.

Table 9.3: Data instance characteristics

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

9.6.2 Computational Results

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

Results from devised Branch-and-Price method

The results from our developed Branch-and-Price method are obtained using a PC with 4.0 GB RAM and an Intel(R) Core(TM)2 Duo CPU P8600, 2.4 GHz processor under a 64 bit Windows 7. The algorithm is implemented in C++ using Cplex 12.4 with default settings to solve the master problem and the time window branching parameter ϵ set to 0.001, which corresponds to 1.44 minutes on our data instances. Table 9.4 shows the results from running our algorithm on the 16 data instances. The column 'Inst' gives the instance number while columns 'Objective' and 'Gap' contains, respectively, the objective value of the best solution found and the integrality gap in percentage for this solution. In column 'Time' we list the time used to solve the problem. We allow a maximum running time of 3600 seconds and if the algorithm runs out of time before closing the integrality gap, we list the time it took to find the best solution and put a '*' to indicate that we ran out of time. Column 'Vars' lists the number of variables in the final master problem, i.e. the number of columns generated. Column 'Nodes' gives the number of explored Branch-and-Bound nodes and, finally, in columns 'Time_{LP}' and 'Time_{Sub}' we have the time spent solving, respectively, the master problem and the subproblems.

Table 9.4: Results from Branch-and-Price method

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

From Table 9.4 we first see that on four instances the algorithm is unable to close the integrality gap after 3600 seconds. However, all four integrality gaps are well below 1% and hence very small. Next, we see that, aside from instance 16, the algorithm very quickly finds a good if not optimal solution. On instance 16, the time to find the best solution is however 1046 seconds. Looking at the number of generated variables, we see that it ranges from as low as just 231 variables up to 73,059. The number of Branch-and-Bound nodes shows a similar diversity of the test instances as this number ranges from just 1 node, i.e. problem solved at root node, up to as many as 175,184 nodes. Combining the number of variables and the number of nodes with the time usage for master problem and subproblems, we note that the four occasions where the algorithm runs out of time can be divided into two different categories. For problem instances 10 and 16, so many variables are generated that the master problem becomes so large, that the computation time to solve it

repeatedly dominates the procedure. For these instances, it would probably help to use a higher value for the time window branching parameter ϵ. This will in general speed up the algorithm; however, it can sacrifice optimality. On the other hand, for instances 14 and 15, the number of generated variables is not very high. Instead, the number of explored nodes is huge and suggests that for these instances it would help to implement a different search strategy for the branch-and-bound tree, e.g. Best-First-Search. It could also help to use different selection rules for time windows and split times in the time window branching scheme, or for ship-voyage pairs in the constraint branching scheme. More generally, the branching scheme might be improved by using strong branching (see Achterberg et al. (2005)). Overall we note that our algorithm is able to find very good if not optimal solutions extremely fast, though one instance requires longer time.

As already mentioned, Norstad et al. (2013) find that voyage separation requirements can significantly improve the spread of the voyages and at only marginal profit reductions. Although we do not wish to repeat their analysis here, we note that we arrive at similar findings after running the 16 instances again without voyage separation requirements. In fact, the profit reduction is below 1% on all 16 instances. The complexity added from the voyage separation requirements is however not insignificant and this is most notable for instance 10 where we ran out of time when voyage separation requirements were included in the problem. Running instance 10 again, though this time without these separation requirements, we are able to solve the instance to optimality using just 0.12 seconds and exploring just 7 nodes. On instance 16 we still run out of time but now the best solution is found after just 535 seconds, where it was 1046 seconds with the separation requirements included.

Results from A Priori Path Generation method

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

The authors from (Norstad et al., 2013) have provided us with results from running their APPG method on the same 16 data instances that we use here. Their results are obtained using a comparable DELL Latitude Laptop with Intel Core i5 CPU (4x2.40 GHz), 4GB DDR2 RM running on Windows 7. Their path generator is implemented in C# while the path flow model is solved with Xpress MP 7.0 64 bit. Results from running their method also with a time limit of 3600 seconds on the 16 data instances considered here are given in Table 9.5. The column headers are the same as in Table 9.4 though we have added a column denoted 'Constrs' which lists the number of constraints in the path flow model.

Ignoring the objective function values for now, we first note from the 'Time' column in Table 9.5 that the time usage from this method is generally a lot longer than from our algorithm. However, this method never runs out of time and therefore never exhibits any integrality gaps. It would therefore be natural to conclude that the APPG method is slower but more stable than our method. However, although Norstad et al. (2013) present their APPG method as an exact one, we claim that this is not the case. We discuss this further in the next section when comparing results from the APPG method and our Branch-and-Price method. For now we focus on the APPG method itself and further investigate the actual implementation of it. From Norstad et al. (2013) we have the following quote describing the details of their implementation:

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

As discussed in Section 9.4, without the voyage separation requirements it is sufficient to include the profit maximising schedule for each ship and cargo set. However, with the separation

Table 9.5: Results from A Priori Path Generation method

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

requirements included in the problem this is no longer true since the actual timing of port calls in a schedule for one ship can affect the timing of port calls for schedules of other ships. Instead, any feasible schedule for a ship can potentially be part of the optimal solution and the master problem, or in their case the path flow formulation, can contain several schedules all corresponding to the same set of voyage and maintenance stops. Therefore, the 'simple dominance test' mentioned in the quote above actually sacrifices optimality of the APPG method as implemented by Norstad et al. (2013). With the voyage separation requirements included the a priori generation approach should be quite time consuming which is why we turned to dynamic column generation. However, because of this dominance test the APPG method becomes manageable even with these separation requirements included. So, this dominance test makes their APPG implementation extremely efficient, though it does sacrifice optimality.

Comparing the two methods

To properly evaluate our devised Branch-and-Price method we use this section to compare it with the APPG method from Norstad et al. (2013). Table 9.6 therefore summarises the key values from Tables 9.4 and 9.5 from running each of these two methods on the considered 16 data instances. For the APPG method we have added a column denoted 'Gap' which contains the integrality gap in percentage from comparing the APPG solution with the bound obtained from our Branch-and-Price method. Furthermore, the last column denoted 'Obj. Incr.' lists the percentage increase in objective function value from using our algorithm compared to the APPG algorithm.

From Table 9.6 we first note that, compared to the bound obtained from our algorithm, the APPG method now experiences integrality gaps for 9 out of the 16 instances. We also note that these gaps are consistently larger than our gaps which is consistent with the fact that our solutions are consistently equal to or better than the ones obtained from the APPG method. In fact, the profit increase from using our algorithm compared to the APPG method is as high as 6% for one instance. Focusing instead on the time usage of the two algorithms, we see that on the instances where our algorithm does not run out of time, the APPG method consistently uses the same or longer time. On the four instances where our algorithm runs out time, we manage to find a better solution than the APPG method and for three out of these four instances this better solution is found much faster than the solution from the APPG method. However, for instance 16 this is not true. Finally, we turn to the variable count for each algorithm and note that, aside from instances 10 and 16 where our algorithm runs out time due to an extensive generation of variables, the APPG

Table 9.6: Comparing solution methods

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

method consistently includes much more variables than our method. This is consistent with our dynamic column generation approach which only generates columns as needed.

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

9.7 Concluding Remarks

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

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

Running our algorithm on all 16 instances without the voyage separation requirements showed that the profit reduction from including the separation requirements is below 1% on all instances. This is consistent with the findings in Norstad et al. (2013) who conclude that voyage separation requirements can significantly improve the spread of the voyages and at only marginal profit reductions. There is however a significant increase in problem complexity from these additional

requirements and for one instance this increase caused the algorithm to require more than 3600 seconds to solve a problem that was otherwise solvable within just 0.12 seconds without the additional requirements.

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

Overall we have developed a new, exact method for the tramp ship routing and scheduling problem with voyage separation requirements. This method is extremely fast at finding very good if not optimal solutions, although one instance requires longer time. An interesting extension of this work would be to explore different branch-and-bound search procedures as well as different modifications to the branching schemes. Finally, it would also be interesting to investigate the effect of modifying the APPG method to obtain optimal solutions, i.e. removing the dominance test discussed in Section 9.6.2.

Acknowledgements

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

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In tramp shipping, ships operate much like taxies, following the available demand. For tramp operators, a main concern is efficient and continuous planning of routes and schedules for individual ships. This thesis therefore aims at developing new mathematical models and solution methods for tramp ship routing and scheduling problems. This is done in the context of Operations Research. The first part of the thesis contains a comprehensive introduction to tramp ship routing and scheduling including an analysis of the current status and future direction of research within tramp ship routing and scheduling. We argue that rather than developing new solution methods for the basic routing and scheduling problem, focus should now be on extending this basic problem to include additional complexities and develop suitable solution methods for those extensions. The second part of the thesis therefore explores three distinct ways of extending the basic tramp ship routing and scheduling problem to include additional complexities. First, we explore the integration of bunker planning, then we discuss a possible method for incorporating tank allocations and finally, we consider the inclusion of voyage separation requirements.

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk