Maritime Inventory Routing using Constraint Programming

by

J.T. Teitsma

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on 25th of September 2020.

This research was undertaken in partial fulfillment of the Master's Applied Mathematics. ORTEC B.V. has agreed to supervise and support this research, in addition to supervision from TU Delft.

Student number:	4327454	
Thesis committee:	K.I. Aardal,	Faculty of Electrical Engineering, Mathematics and Computer Science, TU Delft, supervisor
	R.B.O. Kerkkamp,	ORTEC B.V., supervisor
	M.M. de Weerdt,	Faculty of Electrical Engineering, Mathematics and Computer Science, TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

ORTEC OPTIMIZE YOUR WORLD

Abstract

The maritime inventory routing problem (MIRP) is a tactical and operational planning problem, that takes an integrated view on ship scheduling and inventory management for bulk products. Given production and consumption levels during a predetermined planning horizon, the problem aims at finding delivery schedules with minimal travel costs, such that the inventory bounds at both production and consumption ports are satisfied during the entire planning horizon.

In this thesis, we consider instances of the MIRP with a heterogenous fleet and multiple products. We formulate both a mixed integer programming (MIP) and constraint programming (CP) model for these instances. These models are solved using commercially available dedicated solvers for both formalism. For small instance sizes and short planning periods, the MIP approach is prevalent in finding solutions quickly and of good quality. Due to the inventory constraints in the problem however, the MIP approach suffers from scalability issues more heavily than the CP approach. A rolling-horizon heuristic is proposed in order to find solutions of better quality in comparable running time.

Contents

1 Introduction 1
- 1.1 Shipping industry . . . 1
- 1.2 Motivation . . . 3
- 1.3 Scope . . . 3
- 1.4 Research questions . . . 4
- 1.5 Outline of this thesis . . . 5

2 Maritime inventory routing 7
- 2.1 Background. . . . 8
- 2.2 Classification of literature . . . 10
- 2.3 Solution approaches . . . 15

3 Mixed integer programming 19
- 3.1 Background. . . . 19
- 3.2 Branch and bound . . . 20

4 Constraint programming 23
- 4.1 Theoretical background . . . 23
- 4.2 Constraint-based scheduling . . . 27

5 Problem description 39

6 Model description 43
- 6.1 Contribution . . . 43
- 6.2 Mixed integer programming model . . . 45
- 6.3 Constraint programming model . . . 54
- 6.4 Comparison between MIP and CP model . . . 63

7 Computational study 67
- 7.1 Data description . . . 67
- 7.2 Instances . . . 68
- 7.3 Exact solution methods . . . 70
- 7.4 Rolling-horizon heuristic . . . 79

8 Discussion 85

A Model formulations 89
A.1 Mixed integer programming model 89
A.2 Constraint programming model 91

1

Introduction

This thesis is about solution approaches for the maritime inventory routing problem. In this chapter, the topic is introduced by placing this problem in the context of the industrial shipping industry. Subsequently, we provide the motivation and scope of our study and give the outline of this thesis.

1.1. Shipping industry

Shipping transportation is an important mode of transport in international trade. In comparison to other modes of transport, ships have the advantage of being cost efficient for moving large cargoes over long distances (Rodrigue et al., 2016). The United Nations Conference on Trade And Development provide annual reports on the maritime transport industry (UNCTAD, 2019). According to this report, the total amount of loaded cargo was 11.0 billion tons in 2018, which corresponds to an average yearly increase of 3.9% between the years 1970 and 2018. The total development of the maritime shipping industry during these years is shown in Figure 1.1.

1.1.1. Modes of transport

Maritime transportation can be divided into three modes of operation, namely: liner shipping, tramp shipping and industrial shipping. Christiansen and Fagerholt (2009) elaborate on these three types. Liner shipping is the transportation of goods using ships that have fixed routes in a published schedule. Examples of this mode of transportation include container shipping. More recently, the use of roll-on-roll-off ships, used to transport cars or other vehicles, has expanded within liner shipping. Next to that, tramp shipping concerns the transportation of a combination of fixed contract cargoes and optional

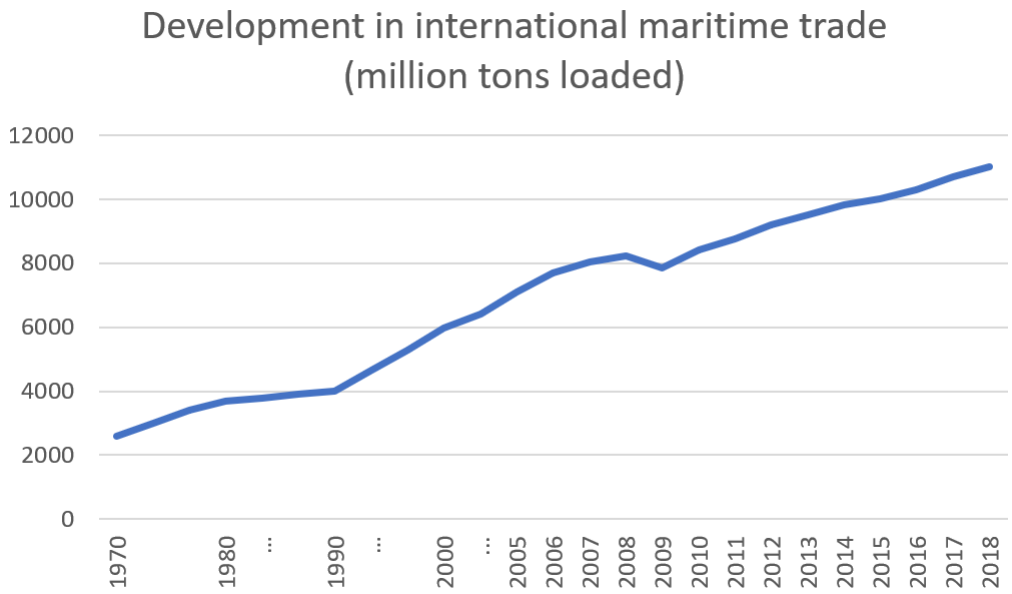

Figure 1.1: The development of maritime shipping during the years 1970-2018, in million tonnes loaded (UNCTAD, 2019).

spot cargoes. The mandatory cargoes come from contracts between the shipping company and cargo owners. The challenge is to maximize profit by transporting additional spot cargoes and constructing schedules accordingly. In industrial shipping, a shipper controls a fleet of ships and must transport cargoes to meet demand at a minimal cost. This mode of transport often occurs at vertically integrated companies, that handle both production and transportation of products.

1.1.2. Planning decisions

Planning decisions in maritime shipping can be made on multiple levels. More specifically, it is possible to identify decisions on strategical, tactical and operational levels (Christiansen and Fagerholt, 2009). Strategical decisions are long-term decisions, such as fleet composition, port location or network design problems. On a tactical level, shipping companies need to decide on problems with a much shorter time horizon, typically ranging from one week to one year. Problems with a tactical aim are for example scheduling and inventory management problems. Lastly, operational planning concerns the decisions with a planning horizon up to one week. Decisions that are made on this level include speed selection and ship loading during port operations. Ship routing decisions are made on both a tactical and an operational level.

1.1.3. Maritime inventory routing

A maritime inventory routing problem is an industrial shipping problem that concerns both the planning and scheduling of ships as well as inventory management at the ports within an maritime transportation network. These decisions are often made on an tactical level. Chapter 2 contains a review of scientific research that is related to the maritime inventory routing problem of interest in this thesis. In Chapter 5, a precise problem definition is presented.

1.2. Motivation

Over the recent years, constraint programming (CP) has gained interest in the operations research community. Especially scheduling is one of the most promising application area of CP, as Laborie et al. (2018) points out. There are some important reasons why the application of CP to scheduling problems has been successful. First of all, the availability of global constraints such as the `cumulative` and the `disjunctive` are very well suited to model the use of resources in this kind of problems. Next to that, CP can be effective when there exist precedence relations between activities in the problem, as the constraint propagation procedures for these kind of relations have shown promise in finding solutions that satisfy these relations. Additionally, the growing interest in the field of constraint programming has fuelled the development of powerful solvers for these kind of problems, which has enabled constraint programming solution approaches to be competitive to mixed-integer programming (MIP) approaches, as shown for example by Ham et al. (2016) and Laborie (2018).

There are many industrial applications that use constraint programming as a solution method, for example in manufacturing (Gedik et al., 2016), civil engineering (Roofigari-Esfahan et al., 2017) and computer scheduling (Gregory et al., 2016). Next to that, transportation problems have also been modelled as scheduling problems, in order to use a constraint programming approach to solve such problems. More specifically, Goel, Slusky et al. (2015) and Giles et al. (2016) have use CP approaches to solve maritime inventory routing problems. Both papers find that constraint programming is able to outperform a MIP approaches for certain instances, motivating the use of constraint programming for our problem.

1.3. Scope

In this thesis, we consider a maritime inventory routing problem for multiple products. This problem has attracted increasing interest from the academic community, as is outlined in the literature review in Chapter 2. The main purpose of this thesis is to compare

the use of constraint programming and mixed integer programming when applied to the maritime inventory routing problem. The scope of this research is discussed in the following sections.

1.3.1. Model formulation

Since the formulation of the maritime inventory routing as an optimization problem by Christiansen and Nygreen (1998), mixed integer programming (MIP) formulations have prevailed as the prime way to describe such problems. In recent years, constraint programming (CP) has developed as an alternative for mixed integer formulations in the field of operations research.

In order to compare MIP and CP for this problem, we develop alternative model formulations following the formalisms of MIP and CP. The mixed integer programming model is based on models that are available in the literature. To our knowledge, the constraint programming model is the first of this kind for maritime inventory problems with multiple products. It is based on models for slightly different but comparable problems (Goel, Slusky et al. (2015) and Giles et al. (2016)).

1.3.2. Solution approach

In order to directly compare the two approaches, we use the built-in solution algorithms in commercial solver software packages for both the mixed integer programming and the constraint programming formulation of the problem. We perform this comparison on problem instances of various sizes, in order to gain an insight in the scalability of both formulations.

Earlier research has shown that the aforementioned direct solution methods often do not succeed in finding good quality solutions for large instance sizes. Therefore, many solution approaches have been proposed in the literature, including decompostion methods, large neighbourhood search and various problem-specific heuristics. In their literature review, Papageorgiou, Cheon et al. (2018) point out that rolling-horizon heuristics seem to be the most promising approach to finding good quality solutions for large instance sizes.

We develop a rolling-horizon heuristic that is based on either constraint programming or mixed integer programming. The MIP and CP approaches are then compared in their ability to find good quality solutions in a heuristic setting.

1.4. Research questions

The scope of this research is summarized in the following research questions:

1. How can the maritime inventory routing problem be modelled using mixed integer programming and constraint programming?

- What design choices need to be made in the formulation of these models?
- What are the (dis)advantages of using MIP and CP?

2. How does the performance of the solution algorithms in commercial solver software for the MIP and CP formulations of the maritime inventory routing problem compare?

   - How does this scale with instance size, for example with the time horizon or number of ports and products?

3. How does the performance of the rolling-horizon heuristics for the MIP and CP formulations of the maritime inventory routing problem compare?

   - How does the use of a rolling-horizon heuristic compare to the solution algorithms of commercial solver software?

1.5. Outline of this thesis

We organize this thesis as follows. Firstly, in Chapter 2, we conduct a literature study of articles concerning the maritime inventory routing problem. We give a historical overview of the development of the research field and classify the most relevant articles based on several characteristics. Secondly, in Chapter 4, we introduce constraint programming to the reader. This chapter consists of a general introduction in constraint programming and a more specific explanation of constraint programming for scheduling problems. In Chapter 5, we give a mathematical description of the maritime inventory routing problem. The two formulations for this problem are proposed in Chapter 6, in which we also discuss their similarities and differences. Next, in Chapter 7, we present our computational study into the performances of the solution methods associated with the MIP and CP model formulations. In this chapter, we also describe the rolling-horizon construction heuristics and present the computational results. Finally, in Chapter 8, we discuss the findings of this thesis and present recommendations for future work.

2

Maritime inventory routing

As outlined in Chapter 1, this thesis is focused on the maritime inventory routing problem (MIRP). The research field concerning this problem has received an increasing amount of attention in recent years. The purpose of this chapter is to familiarize the reader with the problem and describe the developments what developments have been made in this field. We present and discuss the most relevant articles in this chapter.

The goal is to familiarize the reader with the topic and describe the developments in recent years have been.

This literature review consists of three parts. Firstly, in Section 2.1, we discuss the chronological development of the field of maritime inventory routing into the research area that it is today. Secondly, in Section 2.2 we classify the most relevant articles based on several characteristics. These characteristics are subdivided into three categories, the problem, the model and the solution approach. In this way, the reader becomes familiar with the different forms in which maritime inventory routing appears in the literature. In this section, we also discuss the formulation of two constraint programming models as an alternative to mixed integer programming. Later, in the problem description and model description in Chapters 5 and 6, we try to position our paper in a similar manner. Lastly, in Section 2.3, we describe the solution methods that are used in the same articles that we discussed before. The results of Sections 2.2 and 2.3 are summarized in two tables at the end of this chapter.

The purpose of this literature review is to give the reader an overview of the articles that are most related to this thesis. Two extensive literature surveys on MIRPs are Christiansen, Fagerholt, Nygreen et al. (2013) and Andersson et al. (2010). Papageorgiou, Cheon et al. (2018) give a shorter but more recent literature survey, which is mainly focused on the use of matheuristics as a solution approach.

2.1. Background

Maritime inventory routings are a combination of ship scheduling and inventory management, as explained in Chapter 1. They are part of the more general class of inventory routing problems. Bell et al. (1983) were the first to present the integrated view as a variant of the vehicle routing problem that takes inventory costs into account. Dror et al. (1987) defined the general inventory routing problem as a distribution problem in which consumers of a product hold a local inventory and consume a certain quantity every day. The objective is to minimize the costs of the transportation of the product from supplier(s) to customers, while keeping all inventory levels between their lower and upper bounds. This problem arises in situations where one actor is responsible for both the inventory as well as the scheduling of transportation vehicles (Christiansen, Fagerholt, Nygreen et al., 2013).

In the years after its introduction by Bell et al. (1983), the inventory routing problem has been described and studied in many variants and with applications across industries. The scope of inventory routing problems mainly concentrates on road-based and maritime transportation of goods. In their literature review, Andersson et al. (2010) did not find examples of inventory routing based on rail- or air transportation.

Road-based transportation problems have attracted the most attention within the inventory routing literature (Andersson et al., 2010), with applications across many industries. Coelho et al. (2015) provide some examples, such as the transportation of liquid gas (Campbell et al., 2004), automotive components (Richey et al., 2007), groceries (Custódio et al., 2006), fuel (Popovic et al., 2012) and livestock (Oppen et al., 2010). For comprehensive literature reviews on the inventory routing problem in general we refer to Andersson et al. (2010) and Coelho et al. (2015).

MIRPs often have similar distinctive characteristics, that makes them different from their road-based counterparts. Andersson et al. (2010) mention some important differences, of which we repeat two here. Firstly, there is a difference in planning horizons, which are generally longer in the maritime sector because of long travel times and lengthy port operations. Related to this, the time granularity is often rougher for maritime problems. Next to that, the quantities that are loaded and unloaded in road-based applications are often small. In contrast, large quantities, relative to the capacity of a ship, are often loaded and unloaded in the maritime context. Lastly, an industrial fleet of ships is often heterogeneous, because of the long-term investments that are needed to purchase a ship. In contrast to road-based inventory routing papers, where the fleet is assumed to be homogeneous more often, the papers that we included in this review deal with a heterogeneous fleet. This means that ships may differ in size, sailing speed, operating costs etcetera.

Because of these distinctive characteristics, MIRPs have attracted interest from the academic community on their own. The first MIRP was introduced by Miller (1987). Christi-

ansen and Nygreen (1998) present a combined inventory and routing problem with time windows that was based on a maritime case study. In their paper, they formulate a transportation problem for a single product (ammonia) as a mixed integer linear program. Consumption and production rates are assumed to be constant. The column-generating approach that they used became known as the *path-flow* formulation. In a later article, Christiansen (1999) introduced an alternative formulation for this problem as a network model. The nodes correspond to possible events in the planning horizon, namely port visits that a ship can make. This event-based formulation, called *arc-flow*, became widely used in literature. We will discuss the differences between the two formulations in more detail in Section 2.2.1.

The aforementioned models by Christiansen use a continuous variable to keep track of time between port stops, which is an accurate representation of time. Ronen (2002) was the first to develop a discrete-time model for the maritime inventory routing problem. In this model, the planning horizon was subdivided into smaller time periods. This approximation can be made for problems with larger travel times and operation times, such as the MIRP. The use of discrete-time models has become more common over time, with recent papers on the maritime inventory routing problem, such as Goel, Slusky et al. (2015), Agra, Christiansen et al. (2016) and Foss et al. (2016), presenting a model in discrete time. The trade-off between continuous time and discrete time is discussed in more detail in Section 2.2.1.

The MIRP has occurred in many application areas concerning transportation of liquid bulk products. The first papers that related to the maritime inventory routing addressed a transportation problem for ammonia. Next to that, most of the articles find an application for petroleum products, such as Song et al. (2013). In recent years, attention for the simultaneous transportation of multiple products has grown, especially with application in the petroleum and chemistry industries. Al-Khayyal et al. (2007) study the transportation of petrochemicals, whereas Persson et al. (2005) consider bitumen products. Examples of other application areas are the cement industry (Christiansen, Fagerholt, Flatberg et al., 2011), pulp (Andersson, 2011) and calcium carbonate slurry (Dauzère-Pérès et al., 2007).

Over recent years, another application area has grown more interest on its own, namely that of liquefied natural gas (LNG). World-wide demand of LNG accounts for a steadily growing share of the natural gas market (Goel, Slusky et al., 2015). According to Christiansen and Fagerholt (2009), this has resulted in increasing interest for optimal supply chain management for this product. The LNG inventory routing problem is similar to other inventory routing problems, with few additional requirements that are specific to the nature of LNG transport. It was first presented by Grønhaug et al. (2009) and later solved for large instances by Goel, Furman et al. (2012).

Like the first paper on maritime inventory routing by Christiansen and Nygreen (1998), most articles modelled this problem as a mixed integer program. With the development of constraint programming as an alternative to mixed integer programming, another class of models for transportation problems have arisen in recent years. Goel, Slusky et al. (2015) formulated a constraint programming model for the LNG inventory routing problem and introduced a CP-based solution technique. Later, Giles et al. (2016) formulated a different constraint programming model for a supply-and-delivery problem for an oil company. We discuss these constraint programming models in the next section.

Another step in the growing maturity of the research field, was the availability of public data sets. Papageorgiou, Nemhauser et al. (2014) present a library (MIRPLib) of test instances for the maritime inventory routing problem. The main goal of the library was to help the field gain maturity and reduce the "barrier to entry" for interested researchers. Next to the database of benchmark instances, Papageorgiou, Nemhauser et al. (2014) provide best known computational results for each instance.

2.2. Classification of literature

In the previous section, we have seen that there are some characteristics that all MIRPs share, such as time scales, quantities (un)loaded and a heterogeneous fleet. Christiansen and Fagerholt (2009) present a basic version of the MIRP in their paper for the transportation of a single product. However, as Andersson et al. (2010) point out, there exists large variability among problems that are described in the maritime inventory routing literature. The reason for these differences is the wide range of assumptions that could be made in order to define a MIRP or to formulate a model.

In this section, we present a framework that classifies articles based on several important characteristics, in order to properly classify the wide array of problems that occur in the literature. This framework is made of several criteria that encompass multiple aspects of a maritime inventory routing. A similar framework was provided by Andersson et al. (2010) consisting of slightly different characteristics. We will describe these characteristics in Section 2.2.1.

The set of characteristics that we use to classify articles is not sufficient to describe all dimensions of every problem and model that is found in the literature in a detailed way. However, they are sufficient to provide a high-level categorization of the most important articles related to maritime inventory routing.

2.2.1. Problem characteristics

In this section, we present several characteristics to classify MIRPs that are studied in the literature. The problem characteristics that we will cover in this section are:

- Formalism
- Time horizon
- Network
- Products
- Fleet
- Inventory
- Production and demand rates
- Scheduling
- Time
- Compartments

Formulation

Most of the articles that are included in this review, use a mixed integer programming formulation of the problem. In the description of the different characteristics of the problem, the discussion mainly concerns MIP formulations of different aspects of the problem.

To the best of our knowledge, there are two articles that use constraint programming (CP) to model this problem in an alternative way. Goel, Slusky et al. (2015) propose two CP models for the LNG inventory routing problem, which they compare with a mixed integer programming model that they formulated earlier (Goel, Furman et al., 2012). The main reason they give for formulating a constraint programming model is that the mixed integer model suffered from scalability issues for larger instances. Giles et al. (2016) present a CP formulation for a supply-delivery problem that is operated by energy company BP.

All models that are developed in these papers make use of the scheduling formalism, which we explain in Chapter 4. In this formalism, it is possible to model a transportation problem as a scheduling problem. There are some differences in the model formulations given in these articles. In the first model that is formulated by Goel, Slusky et al. (2015), the products that that need to be shipped are modelled as possible tasks, where each cargo can be transported by one ship. In the second model, the tasks represent all possible stops that a ship can make at the ports. These visits need to be scheduled in order to create a feasible schedule. Giles et al. (2016) take the perspective of a port, by enumerating all possible visits that can be made to a port.

Time horizon

We subdivide papers based on the time horizon of the instances. Similar to Foss et al. (2016), we recognize horizons up to one month as short, horizons of multiple months as medium and those of one year or longer as long planning horizons. The early papers of Christiansen and Nygreen (1998) and Christiansen (1999) solved instances with short planning horizon. The paper of Al-Khayyal et al. (2007) is mainly focused on operations in the ports, using planning horizons of up to ten days. Thanks to improvements in available computation power and the development of specialized optimization software, later papers address problems with time horizons up to of a couple of months. The instances that are used in Goel, Furman et al. (2012) and Goel, Slusky et al. (2015) even have a planning horizon of up to one year.

Network

The delivery networks of the papers in this review consists of multiple customers that need to receive products. However, the number of suppliers in the network may vary. Andersson et al. (2010) classifies article based on the structure of the network, where there can be a difference between distribution networks consisting of either one or multiple suppliers. Giles et al. (2016) introduce a problem with a slightly different distribution network, that consists of suppliers, plants and customers. This is a generalization of the standard MIRPs, because the plants can be regarded as ports that have both production and consumption of products.

Products

Maritime inventory routing problems in general deal with bulk products, of which we mentioned some applications in Section 2.1. Many of the research that is conducted, such as in Dauzère-Pérès et al. (2007), is driven by business questions from oil companies. This makes the shipping of oil products one of the main application areas of MIRPs. A distinction can be made, based on whether a model accounts for one or multiple products. In the paper of Christiansen and Nygreen (1998), a model is developed for the transportation of one single product. Ronen (2002) was the first to present a scheduling model for multiple products. This model only accounted for the planning of shipments and did not include decisions on delivery schedules for ships. The arc-flow model of Christiansen (1999) was extended to handle multiple products by Al-Khayyal et al. (2007). The LNG inventory routing problem was solved for large instances by Goel, Furman et al. (2012) and Goel, Slusky et al. (2015). Giles et al. (2016) solve a problem that involves conversion of a single base product into multiple end products.

Fleet

The problems in the literature can be distinguished based on the size of a fleet. A large fleet offers more scheduling possibilities and therefore alters the nature of the problem. We classify small-sized (< 5 ships), medium-sized ($5-10$ ships) and large-sized (> 10 ships) fleets. These three categories are all represented in the articles in this review. Many of the problems deal with small-sized fleets, but efforts have been made to solve the problem for fleets of larger sizes, for example by Dauzère-Pérès et al. (2007), Papageorgiou (2012), Goel, Furman et al. (2012) and Goel, Slusky et al. (2015). Ronen (2002) only considers the scheduling of cargoes from and to ports, without ship schedules. Therefore, the size of the fleet is not relevant for this problem.

Inventory

One of the defining elements of the maritime inventory routing problem is the consideration of stock levels at the producers and consumers. In many of the problems, the inventory

bounds are imposed as hard constraints, for example in Christiansen and Nygreen (1998). Sometimes, it might be inevitable that inventory bounds cannot be satisfied, which means that the stock levels are at their lower or upper bounds.

There are two possible inventory policies that determine how these situations are handled. If inventory levels of consumption ports are allowed to be outside the corresponding bounds, this leads to back-ordering. This means that the corresponding demand must be served at a later point in time. Mathematically speaking, this is equivalent to changing hard inventory constraints into soft constraints, while penalizing the back orders in the objective function. This policy is used in Ronen (2002) and Persson et al. (2005). The other inventory policy is referred to as lost sales. Lost sales occur when inventory levels are kept at its respective bounds during a certain time period, while the production or consumption during that period is considered lost. Among others, Goel, Furman et al. (2012), Goel, Slusky et al. (2015) and Giles et al. (2016) model the inventory management in this way. Papageorgiou (2012) uses an equivalent policy to lost sales, by introducing a spot market for products.

Production and demand rates

This thesis is focused around the maritime inventory routing problem with deterministic production and consumption rates, which are completely known at the beginning of the planning horizon. This means, that we do not include stochastic MIRPs.

In practice, production and consumption profiles are not constant during an entire planning horizon. However, assuming that production rates are constant can simplify the model that should be used. This simplification is made by several authors, e.g., Christiansen and Nygreen (1998), Al-Khayyal et al. (2007), Siswanto et al. (2011) and Agra, Christiansen et al. (2013). On the contrary, varying production and consumption profiles, that are used by for example Papageorgiou (2012) and Giles et al. (2016), give a more correct representation of reality. As an exception to other papers, Grønhaug et al. (2009) let the production quantity be a decision variable of the problem.

Scheduling

As MIRPs are concerned with finding optimal ship delivery schedules, scheduling of port visits and intermediate routes is a core element of these problems. Therefore, the way of generating schedules is an essential characteristic of any model in the literature. In the first paper by Christiansen and Nygreen (1998), the authors propose a model in which routes of a ship are generated a priori. Then, binary decision variables indicate whether or not a ship uses such a route or not. This approach later became known as the path-flow formulation.

In later papers, several alternatives to this approach have been proposed. Ronen (2002) formulated a model that only considered the scheduling of shipments between ports, without assigning them to ships or routes. In this way, the scheduling decision is made separately. Dauzère-Pérès et al. (2007) considered a problem where ships would make one round-trip

from a refinery to exactly one consumption port each day, leading to a simplified scheduling decision for ships.

The approach that would become the most used is called the *arc-flow formulation*, which was introduced in the paper by Christiansen (1999). In the arc-flow formulation, a MIRP is formulated as a flow network. The nodes in the network are possible port stops during the planning horizon. The arcs in the network are possible connections between two nodes. The movement of the ships is represented by the flow through the network, by introducing variables that indicate whether a ship uses the arc between two port stops or not. This event-based model formulation has been used in a majority of the papers in the literature. We discuss this formulation in more detail in the model description in Chapter 6.

The arc-flow and path-flow approaches are compared in the paper by Grønhaug et al. (2009). The main difference between the two formulations is that in an arc-flow formulation, all route information such as arrival times and loading quantities are calculated based on the values of the variables in the problem. In a path-flow formulation, all this information is included in the routes that are generated a priori. As Grønhaug et al. (2009) point out, the path-flow formulation suffers from poor scalability issues. All possible routes need to be included upfront, which means that if the number of ports or the time horizon increases, the number of possible routes in the path-flow formulation increases exponentially. In contrast, the number of nodes and arcs in an arc-flow network scales polynomially with the number of ports and the length of the planning horizon. Grønhaug et al. (2009) show that this phenomenon leads to long solution times for the path-flow formulation, even for the LP relaxation.

Papageorgiou (2012) concludes that for complicated MIRPs with multiple split pickups and split deliveries, arc-based formulations are preferred. He gives two main reasons for this, namely that local search heuristics are better able to make small local changes. Next to that, path-based formulations often have to generate many columns. Given the nature the problem we describe and the size of the case study, the use of arc-based formulations are preferred in the formulation of a model in this thesis.

Time

Another characteristic that distinguishes different models in the literature is the handling of time. In this regard, there are two main directions that could be taken. In the first papers with models for maritime inventory routing, a continuous time variable was used. In the path-flow model that was proposed by Christiansen and Nygreen (1998), ship routes are enumerated a priori. In these routes, arrival, operating and travel times take continuous values, just as it would be in a real-world situation. Also the arc-flow models that were developed later, contain continuous time variable to keep track of the time.

Ronen (2002) was the first to present a model that used an alternative interpretation of time, namely discrete time. In a discrete-time model, the planning horizon is subdivided into fixed time periods. An index that represents time is included to all relevant variables and constraints in the model. Persson et al. (2005) presented an arc-flow model that used discrete time.

Although continuous-time formulations were developed first, recently more discrete-time models have been proposed. The biggest advantage of using a discrete-time scheme is that it is easier to deal with varying production and consumption profiles. Agra, Christiansen et al. (2016) formulate both a continuous-time and a discrete-time model and compare the two.

Compartments

The continuous-time arc-flow model by Christiansen (1999) has been extended to contain multiple products by Al-Khayyal et al. (2007). In order to deal with multiple products, it must be determined how to model the way that products are loaded into ships. As most products in practical cases are not mixable, one needs to divide the products over different tanks. It is then a design decision, how to model this allocation of products to tanks mathematically.

In earlier papers by Persson et al. (2005) and Dauzère-Pérès et al. (2007) the allocation of products to the ship was not considered explicitly. Products could be loaded into the ship until the total ship capacity was reached. No further distinctions were made between the products. This approach was later also chosen by Agra, Christiansen et al. (2013). Al-Khayyal et al. (2007) introduced the concept of *dedicated* compartments. They assumed that every ship contained exactly one compartment that could be filled with one specific type of product. This is a simplified representation of reality, as often different types of products could be loaded into a compartment after it is emptied and cleaned. To model this situation, Siswanto et al. (2011) modelled the multi-product problem using undedicated compartments. The model is formulated in continuous time, similar to Christiansen (1999); Al-Khayyal et al. (2007), but has modifications for loading and unloading activities. This prevents that a product can be loaded into a compartment that already contains another product. Foss et al. (2016) use a discrete time model with undedicated compartments and exploit the structure of the problem in order to strengthen the model using valid inequalities.

2.3. Solution approaches

The articles that are included in this review, use various solution approaches to solve the problem. Almost all papers formulate a mathematical program and feed this to a solver,

where the optimization is done using a branch-and-bound algorithm. As the size of an instance of the MIRP can be very large for real-life problems, this direct method often does not suffice in finding the optimal solution. Therefore, the solutions from this direct method are often compared with alternative approaches, that can handle the complexity of the problem.

The first attempt to tackle this problem, by Christiansen and Nygreen (1998) and Christiansen (1999) made use of a Dantzig-Wolfe decomposition of the problem. In this approach, the problem is subdivided in smaller subproblems that find feasible routes for ships and visit sequences for the ports. As ships can be scheduled independently of each other, this decomposition leads to smaller subproblems that can be solved independently from each other.

The arc-flow models are often solved by problem-specific heuristic, for example in Siswanto et al. (2011). This heuristic is aimed at constructing a good solution based on several selection rules. Song et al. (2013) apply a large neighbourhood search to an arc-flow model. In this algorithm, they iteratively improve an initial solution by locally optimizing the problem with all but two ships fixed. Goel, Furman et al. (2012) then build on this heuristic by proposing a simple construction heuristic to find good solution to large-scale MIRP instances.

In a recent study, Papageorgiou, Cheon et al. (2018) present and adapt several variants of matheuristics, which have been a focal point of research in the previous years. Matheuristics are solution methods that use the optimal solutions of (smaller) mathematical programs. The authors compare multiple matheuristics in their computational study and find new best known solutions to several of the unsolved instances in the MIRPLibrary (Papageorgiou, Nemhauser et al., 2014). Papageorgiou, Cheon et al. (2018) point out that matheuristics rely for a large part on the performance of the solver that is used to solve the underlying smaller mixed integer linear programs.

Both Goel, Slusky et al. (2015) and Giles et al. (2016) compare a CP approach to a mixed integer programming approach, by letting commercial solvers solve the two different problem formulations. Next to that, Goel, Slusky et al. (2015) develop both a CP heuristic and a hybrid heuristic, which incorporates elements from both constraint programming and mixed integer programming.

Table 2.1: This table contains an overview of all model characteristics for the articles that are included in this literature review.

				Problem characteristics						
Paper	**Formalism**	**Time horizon**	**Network structure**	**Products**	**Fleet**	**Inventory**	**Rates**	**Time**	**Scheduling**	**Compartments**
Christiansen and Nygreen (1998)	MIP	Short	Multiple	Single	Small	Hard bounds	Constant	Continuous	Path-flow	N/A
Christiansen (1999)	MIP	Short	Multiple	Single	Small	Hard bounds	Constant	Continuous	Arc-flow	N/A
Ronen (2002)	MIP	Short	Single/multiple	Multiple	N/A	Back-order	Varying	Discrete	Shipments	N/A
Persson et al. (2005)	MIP	Short/medium	Multiple	Multiple	Small	Back-order	Varying	Discrete	Arc-flow	No compartments
Dauzère-Pérès et al. (2007)	MIP	Short	Single	Multiple	Large	Hard bounds	Varying	Discrete	Daily trip	No compartments
Al-Khayyal et al. (2007)	MIP	Short	Multiple	Multiple	Small	Hard bounds	Constant	Continuous	Arc-flow	Dedicated
Siswanto et al. (2011)	MIP	Short	Single	Multiple	Small	Hard bounds	Constant	Continuous	Arc-flow	Undedicated
Papageorgiou (2012)	MIP	Medium	Single/multiple	Single	Large	Lost sales	Varying	Discrete	Arc-flow	N/A
Song et al. (2013)	MIP	Medium	Multiple	Single	Medium	Hard bounds	Varying	Discrete	Arc-flow	N/A
Agra, Christiansen et al. (2013)	MIP	Short	Multiple	Multiple	Small	Hard bounds	Varying	Continuous	Arc-flow	Dedicated
Agra, Christiansen et al. (2016)	MIP	Short	Multiple	Multiple	Small	Hard bounds	Constant	Continuous/discrete	Arc-flow	No compartments
Foss et al. (2016)	MIP	Short	Multiple	Multiple	Small	Hard bounds	Varying	Discrete	Arc-flow	Undedicated
Grønhaug et al. (2009)	MIP	Short/medium	Multiple	Single (LNG)	Small	Hard bounds	Decision	Discrete	Arc-flow	N/A
Goel, Furman et al. (2012)	MIP	Long	Single	Single (LNG)	Medium/large	Lost sales	Varying	Discrete	Arc-flow	N/A
Goel, Slusky et al. (2015)	CP	Long	Single	Single (LNG)	Medium/large	Lost sales	Varying	Discrete	CP	N/A
Giles et al. (2016)	CP	Medium	Multiple	Multiple	Medium	Lost sales	Varying	Discrete	CP	Undedicated

Table 2.2: This table contains an overview of all solution approaches in the articles that are included in this literature review.

	Solution approaches								
Paper	**MIP approach**	**CP approach**	**Decomposition**	**Construction heuristic**	**Local search**	**LNS**	**Matheuristic**	**CP heuristic**	**Hybrid heuristic**
Christiansen and Nygreen (1998)	✓		✓						
Christiansen (1999)	✓		✓						
Ronen (2002)	✓			✓					
Persson et al. (2005)	✓			✓					
Dauzère-Pérès et al. (2007)				✓					
Al-Khayyal et al. (2007)	✓								
Siswanto et al. (2011)	✓			✓					
Papageorgiou (2012)	✓			✓	✓				
Song et al. (2013)	✓			✓		✓			
Agra, Christiansen et al. (2013)	✓								
Agra, Christiansen et al. (2016)	✓								
Foss et al. (2016)	✓						✓		
Grønhaug et al. (2009)	✓								
Goel, Furman et al. (2012)	✓			✓		✓			
Goel, Slusky et al. (2015)	✓	✓						✓	✓
Giles et al. (2016)	✓	✓							

3

Mixed integer programming

In this thesis, we compare a mixed integer programming approach to a constraint programming approach for the maritime inventory routing problem. In this chapter, we briefly introduce the principle of mixed integer programming, where we restrict ourselves to *mixed integer linear programming*. Next to that, we explain the branch and bound algorithm, which is used in the solution algorithm that is implemented in the commercial solver that we use to analyse our problem.

Branch and bound is only a single topic in the research field of mixed integer programming, but is highlighted here because of the relevance to our research. For further information and a broader picture of this research field, we refer the reader to textbooks on this topic, for example Conforti et al. (2014).

3.1. Background

The theoretical foundations for the research field in integer programming are motivated by the paper of Dantzig et al. (1954) and Ralph Gomory's work on cutting planes for integer programs.

Suppose we have n-dimensional variable $x = (x_1, \ldots, x_n)$. For subset $I \subseteq \{1, \ldots, n\}$ these variables are integer, whereas for the variables $\{1, \ldots, n\} \setminus I$ the variables are continuous. Then a *mixed integer linear program* is given by:

$$
\begin{aligned}
\text{minimize} \quad & z = c^T x \\
\text{subject to} \quad & Ax \leq b, \\
& l_j \leq x_j \leq u_j, j \in \{1, \ldots, n\}, \\
& x_j \in \mathbb{Z}_{\geq 0}, \qquad j \in I, \\
& x_j \in \mathbb{R}_{\geq 0} \qquad j \in \{1, \ldots, n\} \setminus I.
\end{aligned}
$$

where c is an n-dimensional vector, A an $m \times n$ matrix, and b an m-dimensional vector. The set of feasible solutions for this program is given by the set

$$S = \left\{ x \middle| Ax \leq b,\ x_j \in \mathbb{Z}_{\geq 0} \text{ for } j \in I,\ x_j \in \mathbb{R}_{\geq 0} \text{ for } j \in \{1,\ldots,n\} \setminus I \right\}.$$

A solution x^* to the problem satisfies $c^T x^* \geq c^T x$ for all $x \in S \setminus x^*$.

A *relaxation* of a mixed integer linear problem is a modification of the problem, in which certain requirements are relaxed, meaning that the set of feasible solutions of the relaxed problem is larger than the feasible set of the original problem. One relaxation to a mixed integer linear program, in which all integrality requirements are relaxed, is called the *linear programming* relaxation or LP relaxation.

The set of feasible solutions for this program is given by the polyhedron

$$S = \left\{ x \middle| Ax \leq b,\ x_j \in \mathbb{R}_{\geq 0} \text{ for } j \in \{1,\ldots,n\} right \}.$$

3.2. Branch and bound

One way to solve mixed integer linear programs, would be to enumerate all possible solutions and check which solution is the best one. However, for practical problems, this is a highly ineffective way of solving the problem. Branch and bound is an alternative method that solves a mixed linear programs by splitting up the problem under consideration into sub problems that are easier to solve. By solving the sub problems, one aims to find a solution for the original mixed integer program.

Branch and bound aims at systematically searching the space of all feasible solutions to a mixed integer linear program, while avoiding the need to enumerate all possibilities. In the search process, the space of feasible solutions is iteratively split up in smaller subsets. This process can be represented by a *search tree*. The root of this tree is the original problem, also called the *master problem*. The sub problems that are created during the solution process are represented by *nodes* in this search tree.

The process of splitting a problem up into two sub problems is called *branching*. As explained later, branching can take place in the case that the solution to the LP relaxation of a problem in the search tree does not meet all integrality requirements of the master problem, meaning that it contain at least one of the variables with an integrality requirement in the master problem has a non-integer value in the solution to the LP relaxation. In this case, one of these variables is selected and two sub problems are created. In either sub problem, an additional requirement is imposed on the selected variable, to force the solution away from the fractional solution that was found in the LP relaxation.

The algorithm is initialised using a lower bound and an upper bound to the objective

value. The lower bound is the value of the LP relaxation of the master problem, whereas the upper bound is given by any feasible solution to the problem. In case no feasible solution is known, $+\infty$ is used as an upper bound to the objective value.

After initialisation, the values of the lower and upper bounds are stored during the solution process, and can be updated during single iterations. For each sub problem at each node, a lower bound to the objective value is calculated. In the case of branch and bound methods for mixed integer linear programs, a lower bound can be calculated by solving a linear relaxation of the sub problem at the node in the search tree. In some cases, it is possible to terminate the branching process at a given node. This step, called *pruning*, means that the sub problem at that node is not partitioned later in the process. This can happen in the following cases:

- The LP relaxation is infeasible, because of the sequence of branching choices that leads to this sub problem is infeasible.
- The solution of the LP relaxation satisfies all integrality conditions of the master problem, which means that the LP solution is feasible for the mixed integer program. The node can be pruned and does not need to be partitioned any further.

  In this case, the values of the upper bound to the objective value of the master problem needs to be updated. If the objective value of the newly found solution is lower than the upper bound that was stored, the upper bound is lower to this objective value.
- The lower bound of the linear relaxation is larger than the upper bound that is stored. In this case, the sub problem can never give an integral solution that is better than the solution that is already known, so this node can be pruned.

In case a node cannot be pruned, it needs to be branched, in the way it is explained before. This procedure can be carried out in order to implicitly search the solution space of a mixed integer program.

4

Constraint programming

Constraint Programming (CP) is originally developed as a modelling concept for solving constraint satisfaction problems, where the goal is to find a feasible solution to the problem. As it is possible to write optimisation problems as constraint satisfaction problems, CP gained more attention in the field of operations research, as a method to model and solve combinatorial problems. In particular, CP is a useful method for finding solutions when applied to highly constrained problems, such as scheduling problems.

In Section 4.1, we give a background on constraint programming in general. We discuss important concepts concerning constraint programming, such as the formulation of CP models and search method. Next, in Section 4.2, we introduce a more specific formalism for modelling constraint-based scheduling problems. Again, we describe formulation, inference and the search method for this particular formalism. We use this framework in Chapter 6 to formulate a constraint programming model for the maritime inventory routing problem.

4.1. Theoretical background

A general constraint programming problem can be modelled as a constraint satisfaction problem (CSP). A CSP is defined by a finite set of variables $X = \{x_1, \ldots, x_n\}$, each with a corresponding domain D_i, and a finite set of constraints $C = \{c_1, \ldots, c_m\}$. We will restrict ourselves to constraint satisfaction problems where the variables can take discrete values. Each constraint c_j is expressed as a relation on a number of variables. This set of variables corresponding to constraint c_j is called its *scope* S_j. A solution to a CSP is given by an assignment of values $\hat{x}_i \in D_i$ for each variable x_i, such that all constraints c_j are satisfied. The goal of a CSP can be either to find a feasible solution, to find all possible solutions or to

optimise with respect to an objective function.

4.1.1. Modelling

A constraint satisfaction problem is defined by its variables and constraints. In contrast to mixed integer programs, where variables can be either integer or continuous numbers, variables in a CSP can take more general forms, as long as the domains contain discrete values.

Constraints can have a *declarative* or *procedural* function. In mixed integer programming, *numerical constraints* describe an aspect of the optimisation problem and reduce the solution space of the problem. This is a declarative constraint. It is a distinct characteristic of constraint programming, that constraints are viewed as a procedure that operates on the solution space (Bockmayr et al., 2005). This offers a method to exploit the problem structure in order to find feasible solutions using these procedures.

In order to understand this concept, we consider the following basic example. We have a CSP that contains three variables x_1, x_2 and x_3, with the following domains:

- $D_1 = \{1, 2\}$
- $D_2 = \{1, 3\}$
- $D_3 = \{1, 2, 3\}$

Now we can impose the `allDifferent` constraint on these variables:

$$\texttt{allDifferent}\{x_1, x_2, x_3\} \implies x_1 \neq x_2 \neq x_3.$$

This constraint states that x_1, x_2 and x_3 cannot have the same value in a feasible solution. Next to that, this constraint comes with a procedure to reduce the domains of all variables that are related to this constraint. For example, if x_1 would be set to 1, it would be straightforward to remove 1 from D_2 and D_3.

The possibility to declare custom constraints comes with many advantages. The most important advantage is that this allows for very flexible modelling of a constraint satisfaction problem. Once the structure of the problem is known, it is possible to declare custom constraints by implementing a corresponding procedure. Next to that, we are not restricted to using linear constraints when formulating a problem, which allows for a more intuitive formulation of the problem. As an example, we can look at the `allDifferent` constraint that was introduced above. This constraint can declare the relation between n variables in one single line. On the other hand, it requires $n(n-1)/2$ constraints of the type $x_i \neq x_j$ to formulate the same constraint using numerical expressions.

A related concept in CP is the use of *global constraints*. A global constraint is a high-level constraint class that describes all properties of the individual constraints. When making

use of global constraints, it is only necessary to declare the constraint and the corresponding filtering algorithm (see next section) once, as it will hold for all individual constraints of this type. The `allDifferent` constraint is an example of a global constraint, as it needs to be defined once, but can be use multiple times in one problem formulation.

Up until now, many types of constraints have been defined for known combinatorial structures. A catalogue of all constraints that have been proposed in the literature is kept in the *Global Constraint Catalog* (Demassey, n.d.). In Pesant (2014) an overview of several categories of constraints is given. We will give some examples of constraints related to the scheduling-based formalism in Section 4.2.

4.1.2. Search methods

For a typical constraint satisfaction problem, a constraint programming search method consists of a couple of steps. In this section, we describe the common steps that are included in such a procedure.

Constraint propagation

Each type of constraint is accompanied by a procedure, which is called the *filter* or *propagator*. Remember that a constraint c_j has scope $S_j \subseteq X$, where variables $x_i \in S_j$ have discrete domains D_i. A propagator for constraint c_j reduces the size of the solution space in each iteration, by removing values from domains D_i, that do not satisfy c_j. The application of these procedures is called *constraint propagation*. Constraint propagation is a technique that works directly on the domains of the variables.

The solution process of a constraint programming problem starts with a constraint propagation step based on the initial domains of the variables. In this way, it is possible to remove variables from the original domain, that do not have the desired level of consistency. In general, these techniques are kept to a low computational time complexity, as these procedures are called at every iteration of a solution process. After the constraint propagation phase, three possible outcomes are possible. The first case is that one of the domains becomes empty, meaning that the problem does not have a feasible solution. Secondly, if the domain of every variable consists of exactly one value, then this is the sole solution to the problem. The last possibility is that every domain D_i is non-empty and at least one domain contains multiple values.

When considering constraint propagation in a constraint satisfaction problem, it is important to take the concept of *local consistency* into account. When applying constraint propagation methods to a constraint propagation problem, the goal is to achieve local consistency, which is defined as a property that characterizes necessary conditions on the values that belong to a solution (Bessiere, 2006).

One example of local consistency is *domain consistency*, as defined by Mohr et al. (1988).

A constraint c_j with scope S_j is said to be *arc consistent* if the following holds: for every value $\hat{x}_i \in D_i$ for a variable $x_i \in S_j$, there exists a value $\hat{x}_k \in D_k$ for all variables $x_k \neq x_i \in S_j$. In the case that one variable $x_i \in S_j$ is fixed to value $\hat{x}_i$, but that it is not possible to satisfy a constraint $c_j \in C$, we state that value $\hat{x}_i$ does not have *support*. There exist other, weaker, levels of consistency that can be pursued when it is too computationally expensive to aim for domain consistency. For example, *domain-*($\mathbb{R}$) *consistency* or *bounds-*($\mathbb{R}$) *consistency* (defined by Puget (1998)) do not require values to be in the discrete domains, but on their continuous relaxations, as Pesant (2014) explains.

Search tree

When all domains of the variables in a problem are non-empty and at least one domain is not a singleton, it is required to use a search procedure to find a solution to the problem. Usually the search method involves the construction of a branching tree, where at each step of the process, a variable is fixed to a value in the first branch and that this value is removed from the corresponding domain in the other branch. At every node of the search tree, the domains of the variables are affected, which means that the condition of local consistency could be violated. Therefore, the constraint propagation algorithms can again be applied after every branching decision in order to reduce domains even further. This procedure is repeated until a feasible solution to the problem is found, or it is shown that the problem is infeasible.

Selection strategies

There are a couple of decision strategies that underlie the procedure, as described in the previous paragraph. First of all, the *node selection* determines which node is explored further. In CP, the node selection is often done in a depth-first fashion (Pesant, 2014), combined with a backtracking algorithm. This means that at every node in the tree, a variable is fixed to a certain value, followed by another variable in the subsequent iteration. Whenever this procedure leads to a infeasible solution, at least one variable is uninstantiated and fixed to a different value.

Central elements of the backtracking strategy are the strategies that determine the selection of the next variable to branch on. A straightforward strategy is to use the *fail-first principle*, which aims to find a fail sooner rather than later. This strategy can be implemented by choosing the variable with the smallest domain. The main reason to opt for this strategy is, that in a depth-first strategy it is important to recover from a bad branching decision as quickly as possible. Many other, more sophisticated variable selection heuristics include: weighted degree, impact-based search and activity-based search.

When a variable selection is made, it is in theory possible to enumerate all possible values of this variable. In practice however, it is more common to branch in a binary fashion, which means that the variable is fixed to one value in the first branch and that this value is

removed from its domain in the other branch. *Value-selection* heuristics determine to what value the chosen variable should be fixed. Again, there are many possible ways to choose a value. One notable strategy is the *succeed-fist principle*, that aims to find the value that causes the smallest reduction of the domains, in order to find a feasible solution as soon as possible.

4.2. Constraint-based scheduling

The framework described in the previous section holds for general constraint programming problems. A more specific framework is that of *constraint-based scheduling*, the discipline that studies how to solve scheduling problems using constraint programming. The use of constraint programming can enhance the exploitation of structure of scheduling problems. Baptiste et al. (2006) name two strengths of constraint programming, namely the natural and flexible modelling of scheduling problems as constraint satisfaction problems and the powerful propagation of temporal and resource constraints.

Industrial routing problems such as the problem of interest in this thesis, can also be modelled as constraint-based scheduling problems. In this case, a visit to a location is modelled as a task that needs to be scheduled. It is found by several authors (Goel, Slusky et al. (2015) and Giles et al. (2016)), that solution approaches based on this constraint-based formulation can offer a speed-up relative to mixed-integer-programming approaches. For this reason, we will also use this approach to develop a CP model. The design of our model is explained in Section 6.3.

4.2.1. Modelling

The modelling of constraint-based scheduling problems makes use of the formalism of conditional intervals that is introduced in Laborie and Rogerie (2008) and Laborie, Rogerie et al. (2009). As explained by Laborie (2009), this formalism extends classical constraint programming by introducing new mathematical concepts to capture the structure of scheduling problems. In this section, we will explain three important modelling concepts of this formalism, i.e., *optional interval variables*, *high-level interval variables*, *sequence variables* and *cumulative functions*. Next to that, we describe useful constraints and illustrate the introduced concepts using the example of a job-shop scheduling problem. For the explanation of the mathematical concepts, we follow the lines of Laborie et al. (2018).

Optional interval variables

In this section, we consider a generic scheduling problem, where n tasks need to be scheduled within a finite integer planning horizon T. Because we need to have discrete variables

when using constraint programming, planning horizon T is represented by a discrete set of time points $\mathcal{T} \subseteq [0, T]$. It is common to assume that all time points are integer and have an intermediate distance of exactly 1, i.e., that $\mathcal{T} = \{0, 1, ..., T\}$.

In the CP formalism for scheduling problems, tasks are be represented by *optional interval variables*. For each possible task that can be scheduled during the planning horizon T, we define an an optional interval variable x as a decision variable. The optionality of the variable needs to be emphasized, as x can be either *absent* or *present*. (Laborie, 2009) states that the optionality of interval variables provides a powerful concept for efficiently reasoning with optional or alternative activities.

We denote an absent interval variable that is absent by $\perp$. Present interval variables are half-open intervals $[s, e)$, defined by its start value and end values, s and e respectively. Both s and e are in $\mathcal{T}$, with $s \leq e$. We have a slight abuse of notation by allowing $s = e$. When $s = e$, we represent a zero-length time interval at time s, even though the half-open interval $[s, e)$ is empty in this case. We denote the domain of x by:

$$D(x) = \{\perp\} \cup \{[s, e) \mid s, e \in \mathcal{T},\ s \leq e\}.$$

The presence status of a fixed interval variable x, *presence(x)*, is a binary variable that is `true` when the variable is present and `false` when it is absent. An interval variable is said to be *fixed* if its domain is reduced to a singleton, i.e., when it is either absent ($x = \perp$) or present $\big(x = [s, e)\big)$. The *length* of the interval is defined as $l(x) = e(x) - s(x)$. We say that two interval variables x and y are equal to each other when their intervals coincide or when they are both absent.

In the general definition of conditional interval variables, we regard the start and the end of an interval variable x as variables with $\mathcal{T}$ as their domain. However, it is possible to specify either the start, the end or the length of a variable as fixed upon definition of the variable, by reducing the domain of the corresponding variables. In case of a fixed length $l(x)$, we add an additional constraint which states that $e(x) = s(x) + l(x)$.

It is possible to state that an activity cannot start, end or overlap with a given set of time steps. For this purpose, we first define piecewise-linear function, which are an important concept in this constraint programming formalism. A piecewise-linear function $f : \mathcal{T} \rightarrow \mathbb{Z}_{\geq 0}$ is a function from the set of time points $\mathcal{T}$ to the non-negative integer numbers $\mathbb{Z}_{\geq 0}$. Now let x denote an interval variable and f an non-negative piecewise-linear function. Then we can define the following constraints:

- `forbidStart`(x, f) states that if interval variable x is present, $s(x)$ cannot be equal to t, for all t where $f(t) = 0$,

- `forbidEnd`(x, f) states that if interval variable x is present, $e(x)$ cannot be equal to t, for all t where $f(t-1) = 0$,

- `forbidExtent`(x, f) states that if interval variable x is present, then for all $t \in [s(x), e(x))$, it must hold that $f(t) \neq 0$.

For all constraints above it holds that when x is absent, the constraint is trivially satisfied.

Several numerical expressions can be defined on an interval variable x. The integer expressions *startOf*(x), *endOf*(x) and *lengthOf*(x) return $s(x)$, $e(x)$ and $l(x)$ respectively. Next to that, the numeric expressions *startEval*(x, f), *endEval*(x, f) and *lengthEval*(x), that are defined on an interval variable x and a piecewise-linear function f, evaluate f at $t = s(x)$, $t = e(x)$ and $t = l(x)$ respectively. A value should be specified as an outcome for these expressions in case the interval is absent.

The method *presenceOf*(x) returns the presence status of interval variable x. This attribute can also be used in logical expressions. For example, the logical implication operator $\Rightarrow$ can define a logical relation between two interval variables x and y as follows:

$$presenceOf(x) \Rightarrow presenceOf(y).$$

This expression means that interval variable y must be present whenever x is present. Laborie et al. (2018) argue that binary constraints between the presence of interval variables play a central role in many CP scheduling models, as they are heavily exploited by propagation algorithms.

In addition to logical relations between the presence of interval variables, several *precedence constraints* are available to specify precedence relations between interval variables. Possible precedence relations between optional interval variables x and y are the following:

- `startBeforeStart`(x,y,c) requires that x starts at least c time points before the start of y, i.e., $s(x) \leq s(y) - c$, for c a non-negative integer.
- `startBeforeEnd`(x,y,c): $s(x) \leq e(y) - c$, for c a non-negative integer.
- `endBeforeStart`(x,y,c): $e(x) \leq s(y) - c$, for c a non-negative integer.
- `endBeforeEnd`(x,y,c): $e(x) \leq e(y) - c$, for c a non-negative integer.

Constraints `startAtStart`, `startAtEnd`, `endAtStart` and `endAtEnd` are defined in a similar way, except that the inequality in the precedence relation is replaced by an equality.

High-level interval variables

Next to the optional interval variables that we have introduced in the previous section, the conditional interval formalism allows for constraints over sets of interval variables. As explained by Laborie et al. (2018), the main purpose is to define a hierarchical structure within the model by grouping a set of interval variables into one *high-level* interval variables. The two constraints that we consider are `span` and `alternative`:

- `span`$(x, \{y_1, \ldots, y_n\})$ states that at least one of $\{y_1, \ldots, y_n\}$ is present if and only if x is present. In that case, $s(x)$ is equal to the minimum of $\{s(y_1), \ldots, s(y_n)\}$. Similarly, $e(x)$ is equal to the minimum of $\{s(y_1), \ldots, s(y_n)\}$. In other words, x spans all variables $\{y_1, \ldots, y_n\}$. Interval variable x is absent if and only if $y_1, \ldots, y_n$ are all absent.
- `alternative`$(x, \{y_1, \ldots, y_n\})$ models an exclusive alternative between $\{y_1, \ldots, y_n\}$: x is present if and only if exactly one interval y_i out of the set $\{y_1, \ldots, y_n\}$ is present. In this case, x starts and ends together with this y_i. Interval variable x is absent if and only if $y_1, \ldots, y_n$ are all absent.

Example: Job shop problem

In order to illustrate the concepts that are introduced in this section, we take a look at an example that considers a job shop scheduling problem. We introduce the example here, and again return to this example later in this section.

This problem consists of N jobs j and M machines m. All jobs j consist of multiple operations O_{jm} with processing times $p_{jm} \geq 0$, that each need to be scheduled on the corresponding machine m. Each job j has a completion time C_j; the time at which all operations O_{jm} are completed. For each job j, there exist precedence relations between the operations, that define the order in which the operations need to be completed. An operation cannot start before its predecessor has been finished. The goal of our example is to schedule all operations O_{jm} to the corresponding machines, while minimizing the makespan, i.e., $C_{\max} = \max_{\text{jobs } j}\{C_j\}$.

This problem can be modelled in the conditional interval formalism, by defining interval variables $\{x_{jm} : \text{jobs } j, \text{machines } m\}$ for each operation O_{jm} in the problem. Each interval variable has a fixed length t_{jm} and a variable start- and end time. For each job j the constraint `endBeforeStart`$(x_{jm}, x_{jm'}, \tau)$ captures the precedence relation between two operations from the same job. This formulation ensure that $e(x_{jm}) \leq s(x_{jm'}) - \tau$, where τ is the set-up time between two operations of the same job on different machines. When a machine m has a certain idle time, this can be modelled by defining the corresponding *availability function* F_m and declaring

`forBidExtent`(x_{jm}, F_m) constraints for all interval variables x_{jm} on that machine m.

A more general version of the job shop problem is the *flexible job shop problem*, where certain operations of a job can be assigned to a set of machines instead of just one. A structure like this can be captured by the `alternative`$(x_{jm}, \{x_{jm_1}, \ldots, x_{jm_k}\})$, where $\{jm_1, \ldots, jm_k\}$ are all k job-machine pairs on which operation O_{jm} can be performed. The `alternative` constraint forces the model to pick exactly one of the interval variables $x_{jm_1}, \ldots, x_{jm_k}$ to be present.

Sequence variables

Most of the scheduling problems involve resources that can only perform one task at a time. From the perspective of the resource, a solution can be seen as a sequence of tasks assigned to this resource in a certain order. In order to use this idea in modelling a problem, *sequence variables* are introduced. Sequence variables π are defined on a set of interval variables $\mathcal{X}$. The domain of such a sequence variable π is the set of all possible permutations of $\mathcal{X}$. We denote a permutation of $\mathcal{X}$ by $g(\mathcal{X})$, and the set of permutations by $G(\mathcal{X})$.

We can define the *length* of a permutation $g(\mathcal{X})$ as the number of intervals in $\mathcal{X}$ that are present, i.e.:

$$\mathit{length}(g(\mathcal{X})) = \left| \{x \in \mathcal{X} : \mathit{presenceOf}(x) = \texttt{true}\} \right|.$$

Using this notion of length, we can define a permutation $g(\mathcal{X})$ as a function $g : \mathcal{X} \to \{0, \ldots, |\mathcal{X}|\}$ such that:

- $g(x) = g(y) \iff (x = y) \cup (x = \perp \cap y = \perp)$, for every $x, y \in \mathcal{X}$,
- $g(x) \leq \mathit{length}(g(\mathcal{X}))$, for every $x \in \mathcal{X}$,
- $g(x) = 0 \iff (x = \perp)$, for every $x \in \mathcal{X}$.

The conditions above define $g(\mathcal{X})$ as an ordering of the present variables in $\mathcal{X}$. If x and y are distinct interval variables in $\mathcal{X}$, it can only have the same position in permutation $g(\mathcal{X})$ when they are both absent. In case a variable in $\mathcal{X}$ is absent, its position in $g(\mathcal{X})$ is equal to zero.

Several constraints can be defined on a sequence variable π. First, we mention the constraints that have an impact on the sequencing of interval variables $x \in \mathcal{X}$ in sequence variable π:

- `first`(π, x) states that if an interval variable $x \in \mathcal{X}$ is present, it is the first variable of sequence π. Similarly, `last`(π, x) fixes x to be the last variable in π,

- `before`(π, x, y) states that if both $x \in \mathcal{X}$ and $y \in \mathcal{X}$ are present, then x will appear before y in sequence π,
- `prev`(π, x, y) states that if both $x \in \mathcal{X}$ and $y \in \mathcal{X}$ are present, then x will appear immediately before y in sequence π, such that no other interval variable will be sequenced between x and y.

Next to these sequencing constraints, there are also constraints that are related directly to the start and end times of interval variables. The most important constraint of this kind is the `noOverlap` constraint. In order to properly define this constraint, we first need to introduce two concepts:

- *Type function* Θ: The type function $\Theta : \mathcal{X} \to \{1, \ldots, |\mathcal{X}|\}$ is a function that assigns a *type* to each interval variable x in $\mathcal{X}$. In general, these types can be represented by a non-negative integer number. It is possible to define types for all interval variables in a sequence variable, by initializing a sequence variable as follows:

  $$\pi = \mathit{sequenceVar}(\mathcal{X}, \Theta).$$

  In this notation, type function Θ maps every interval variable $x \in \mathcal{X}$ to a type.
- *Transition distance matrix* $T : \Theta \times \Theta \to \mathbb{Z}_{\geq 0}$ is the minimal time between the end of one interval variable and the start of the next interval variable, depending on the type of both interval variables. The entries of T are non-negative integers.

Let $\pi(\mathcal{X}, \Theta)$ be the sequence variable over the set of interval variables $\mathcal{X}$, with associated type function Θ. Then the `noOverlap` constraint is defined as follows:

$$\texttt{noOverlap}(\pi, T) \iff e(x) + T(\Theta(x), \Theta(y)) \leq s(y),\ \forall x, y \in \mathcal{X}, \text{ s.t. } 0 < g(x) < g(y).$$

As follows from the definition above, the `noOverlap` constraint regulates the transition times between two interval variables $x \in \mathcal{X}$ and $y \in \mathcal{X}$. Because of the requirement that $g(x) < g(y)$, we know that x should be scheduled ahead of y. Additionally, the requirement that both $g(x)$ and $g(y)$ are unequal to zero, tells that both x and y should be present. If this is not the case, the `noOverlap` constraint is trivially satisfied. As T has non-negative integer entries, this constraint also ensures that there cannot be any overlap between interval variables x and y.

A slightly weaker version is the `noOverlapDirect` constraint, that only works on pairs of direct successors, where the `noOverlap` constraint must hold for *all* combinations of variables x and y.

$$\texttt{noOverlapDirect}(\pi, T) \iff e(x) + T(\Theta(x), \Theta(y)) \leq s(y),\ \forall x, y \in \mathcal{X}, \text{ s.t. } 0 < g(x), g(y) = g(x) + 1.$$

Example: Job shop problem (continued)
Sequence variables can be used to model from the perspective of one resource. In this example, we model a sequence variable π_j for each machine in the problem. This variable returns a solution in the form of a permutation of all jobs that need to be scheduled on that machine. Let us also define type function Θ as follows:

$$\Theta(x) = i, \text{ if } x = x_{ij}, \text{ for all } x \in \mathcal{X},$$

where $\mathcal{X}$ is the set of operations x_{ij}. Next to that, T_j represents the setup times between operations for different jobs on machine j.
Then, the `noOverlap`(π_j, T_j) constraint could be used to model transition times between operations that belong to different jobs, for example by changing the setup of a machine. The `noOverlap` constraint ensures that each machine only performs one operation at a time.

We conclude this section on sequence variables with expressions that return different attributes of the successor of an interval variable $x \in \mathcal{X}$ in sequence variable $\pi(\mathcal{X}, \Theta)$. These are the following:

- *typeOfNext*(π, x) returns $\Theta(y)$, where interval variable y is the successor of x in π, i.e., y with $g(y) = g(x) + 1$.
- *startOfNext*(π, x) returns $s(y)$, for y with $g(y) = g(x) + 1$. Similarly, *endOfNext*(π, x) returns $e(y)$ and *lengthOfNext*(π, x) returns $l(y)$, for the same interval variable y.

Additionally, similar expressions *typeOfPrev*(π, x), *startOfPrev*(π, x), *endOfPrev*(π, x) and *lengthOfPrev*(π, x) return the same attributes as the expressions above, but for the interval variable $y \in \mathcal{X}$ that is the predecessor of $x \in \mathcal{X}$ in π, i.e., y with $g(y) = g(x) - 1$.

Cumulative functions
Resources in scheduling functions can be represented by its accumulated usage by activities. A scheduled activity typically increases the usage of a resource at its start and lowers it at its end. In the case of our problem, production increases the level of available products, whereas consumption decreases it. Constraints are imposed on the level of these resources. The levels of the usage of these resources are modelled using *cumulative functions*, also called *cumul functions*.

In order to understand cumul functions, we first describe two *elementary cumul functions*, the *step* and *pulse* functions. At the beginning of this section, we introduced piecewise-

linear functions. The *step* function is one type of a piecewise-linear function, which is often used in this optional interval variable formalism. A step function $f_{\tau,H} : \mathcal{T} \to \mathbb{Z}_{\geq 0}$ is characterized by two integers, a discrete time point $\tau \in \mathcal{T}$ and a non-negative integer H, called its *height*. Given τ and H, $f_{\tau,H}(t)$ is defined as:

$$f_{\tau,H}(t) = \begin{cases} 0, & \text{for } t < \tau, \\ H, & \text{for } t \geq \tau. \end{cases}$$

In the constraint programming formalism, a step function can be defined in relation to a conditional interval variable x. Let x be an interval variable. Then $stepAtEnd(x, H)$ is a step function that has function value H for $t \geq e(x)$ and 0 elsewhere. In other words, $stepAtEnd(x, H)$ is equal to step function $f_{\tau,H}$, where τ equals $e(x)$. Similarly, one can define $stepAtStart(x, H)$ as the step function that has function value H for $t \geq s(x)$ and 0 elsewhere, or $f_{\tau,H}$, where τ equals $s(x)$. A special case is $stepAt(\tau, H)$, which does not depend on an interval variable and is equal to step function $f_{\tau,H}$.

Apart from specifying height H as a constant, the height of a step function can also be a decision variable of the problem. For example, the function $stepAtEnd(x, H_{\min}, H_{\max})$ defines a step function at the end of interval variable x, with variable height H, under the constraints that $H_{\min} \leq H \leq H_{\max}$. This is still a piecewise-constant function, but the height H is not known upon initialization.

In the same way, the functions $stepAtStart(x, H_{\min}, H_{\max})$ and $stepAt(t, H_{\min}, H_{\max})$ are step functions with variable height at the start of interval variable x and time t respectively. For all elementary step functions that are defined in relation to a interval variable x it holds that the function returns the constant zero-function, where $H = 0$, in case x is absent.

The other elementary cumul function is the *pulse* function. A pulse function $f_{x,H} : \mathcal{T} \to \mathbb{Z}_{\geq 0}$ is characterized by an interval variable x and a non-negative integer H, called its *height*. Given x and H, $f_{x,H}(t)$ is defined as:

$$f_{x,H}(t) = \begin{cases} 0, & \text{for } t \leq s(x) \\ H, & \text{for } s(x) \leq t \leq e(x) \\ 0, & \text{for } t \geq e(x). \end{cases}$$

In the constraint programming formalism, a pulse function is called as $pulse(x, H)$. Similar to the step function that was described previously, it is also possible for the *pulse* function to have a variable height. In this case the function is called as $pulse(x, H_{\min}, H_{\max})$, where the height of the pulse function is a decision variable of the problem, which can take values between $H_{\min}$ and $H_{\max}$.

Now that we have introduced the elementary cumul functions, we can define a cumulative function f. A cumulative function f is a composition of elementary cumul functions, i.e., $f : \mathcal{T} \to \mathbb{Z}_{\geq 0}$ is defined as:

$$f = \sum_{i=1}^{N} \epsilon_i f_i, \text{ with } \epsilon_i \in \{-1, +1\}, N \text{ finite and } f_i \text{ an elementary cumul function.}$$

The following constraints can be imposed on cumul function f:

- `alwaysIn`$(f, x, H_{\min}, H_{\max})$ means that f must be in the range $[H_{\min}, H_{\max}]$ during the duration of interval variable x, if x is present. If x is absent, this constraint is automatically satisfied.

- $f \leq h$, for f and h both cumulative functions, states that f cannot take values larger than h. Note that h can also be a constant integer, representing a constant cumulative function. Similarly, one can define $f \geq h$.

The integer expression $heightAtEnd(x, f)$ returns the value of cumulative function f at the end of interval variable x. A similar expression exists in $heightAtStart(x, f)$ for the value of f at the start of interval variable x.

Example: Job shop problem (continued)
Cumulative constraints are used to model the maximum usage of a resource. In a job shop problem, each machine can only perform one operation at a time. We saw that this can be modelled by the `noOverlap` constraint. An alternative way to model this is to define $f_{ij} = pulse(x_{ij}, 1)$ for each operation O_{ij}. Then the cumulative function $f_j = \sum_i f_{ij}$ represents the usage of machine M_j, and the constraint $f_j \leq 1$ models that a machine can only perform one job at a time. Another possible application is that only a fixed number of machines can operate simultaneously, which can modelled in a similar way.

4.2.2. Search algorithm

In order to gain a better understanding of the methods that are used to solve constraint programming, we take a closer look at the automatic search method, which is implemented in the solver that we use, IBM CP Optimizer.

First, we describe the use of presolve methods. Next, we discuss the constraint propagation methods that are implemented. The solver uses two different strategies to explore the

search space, namely large neighbourhood search and failure-directed search. We briefly discuss both strategies at the end of this section.

Presolve

The main goal of the presolve phase, is to automatically improve a model by removing redundancies and poorly formulated constraints. One example is presented in Laborie et al. (2018). The expressions *endOf*(x) <= *startOf*(y) and `endBeforeStart`(x, y) relating to interval variables x and y declare the same, namely a temporal relation between the end of x and the start of y. However, because of a dedicated algorithm, propagation of the `endBeforeStart`(x, y) constraint gives a much better performance. In the presolve phase, such constraints are automatically identified and rewritten.

Other examples, mentioned by Laborie (2013) of available presolve procedures include the aggregation of $\neq$ constraints into an `alternative` constraint, the replacement of often-occurring expressions by a new variable and the elimination of redundant constraints.

Constraint propagation

As described in Section 4.1.2, an important part of constraint programming search algorithms are the propagation methods. In the previous section, we described the constraint-based scheduling formalism using conditional interval. Several constraints that fit in this formalism have been introduced, which can be divided into three categories: *logical constraints*, *temporal constraints* and *resource constraints*. In this section, we discuss the constraint propagation techniques for the constraints in these three categories.

Let the presence status of an interval variable x be represented by the binary literal l_x. All logical constraints between the presences status of interval variables are stored as literals in a logical network, similar to the *implication graph* that is described by Brafman (2001). The nodes of this graph are all literals l_x and their negations $\neg l_x$. For every logical constraint between the presence status of two interval variables x and y of the form

$$l_x \Rightarrow l_y,$$

an edge is added between the corresponding literals in the implication graph. Laborie et al. (2018) point out that there are several objectives for using this implication network. Firstly, it is possible to detect inconsistencies between logical constraints. Next to that, the network provides $O(1)$ access to the logical relation between any two intervals x and y, which is used by the propagation of temporal constraints. Lastly, when a new implication relation between two interval variables x and y is added to the network, the propagation to other interval variables can be started, by assessing the network.

Similar to the logical constraints, temporal constraints, are represented using a tem-

poral network. An example of such a temporal constraint is a precedence relation between interval variables. The construction and use of a temporal network is described in detail by Laborie and Rogerie (2008). The set of nodes $\{p_i\}_i$ of a temporal network that corresponds to a certain model, is equal to the set of all start points and end points of all interval variables x that are included in that model. Remember from Section 4.2 that temporal constraints on interval variables x and y can come in one of the forms, such as `startAtStart`, `endBeforeEnd` or any similar combination. Such a temporal constraint is modelled in a temporal network by adding an arc between the two nodes p_i and p_j. This arc denotes c as the minimum amount of delay between p_i and p_j, if both p_i and p_j are present. Such an arc is denoted as (p_i, p_j, c).

More specifically, let $z(p_i)$ denote the presence status of p_i and let $t(p_i)$ be the (variable) time of point p_i. Then the arc between nodes p_i and p_j denotes the following logical expression:

$$z(p_i) \wedge z(p_j) \Rightarrow t(p_i) \leq t(p_j) - c.$$

As an example, the constraint `endBeforeStart`(x, y, c) is represented by the arc $(e(x), s(y), c)$ and states that $e(x) \leq s(y) - c$ if both x and y are present.

Constraint propagation in this temporal network can make use of implications between the presence statuses of interval variables in the network. When for a given arc (p_i, p_j, c), the presence relation $presenceOf(p_i) \Rightarrow presenceOf(p_j)$ can be inferred from the logical network, the arc can propagate the time bounds for nodes p_i and p_j in the temporal network. This means that time points p_i and p_j do not necessarily need to be present for the propagation to take place, as long as the logical relation is defined.

Multiple methods to perform the propagation in temporal constraint networks are described in Dechter et al. (1991). These methods can also be extended to account for the use of optionality in the model. In CP Optimizer, the initial propagation is performed by a modified version of the Bellman-Ford algorithm (Cherkassky et al., 1996). During the solution process, temporal constraints are propagated again, when for example a time bound changes or a new logical implication can be inferred. This incremental propagation of temporal constraints is performed by CP Optimizer using an extension of the positive cycle algorithm that is proposed by Cesta et al. (1996). The main difference between the original algorithm and the one that is implemented in the solver, is that propagation is only performed for the arcs that are allowed to propagate by the implication relations in the temporal network.

The propagation of resource constraints is done using a timetable algorithm. This algorithm uses the notion of *conflicting rectangles*, which are parts of the schedule that violate the resource constraint, and propagates by removing all conflicting rectangles from

the schedule. A recent implementation of the timetable algorithm is described by Gay et al. (2015). This algorithm has a polynomial running time, making it efficient and powerful during the filtering phase. As the timetable algorithm is not powerful enough to prove optimality of a solution, some other propagation algorithms for edge-finding are included in the solver, that can propagate the resource constraints even more. For example, the algorithm that is provided by Vilím (2011) is an edge-finding algorithm for cumulative resource constraints, such as the `alwaysIn` constraint that was described in the previous section.

Search strategies

Large-neighbourhood search (LNS) is a solution method in CP Optimizer that makes use of successive relaxation and re-optimisation (Laborie et al., 2018). After a first feasible solution is found, a number of iterations are carried out. Each iteration contains a relaxation step and a re-optimisation of the relaxed solution. This process is continued until some predefined condition is met. Typically, this means that either a time limit is reached or that the solution is proved to be optimal.

LNS does not try to prove optimality of a solution by exploring the entire search space. One of the risks is therefore that it gets trapped in a local minimum. A method to prove optimality of a solution is failure-directed search (FDS), which does enables complete exploration of the search space. FDS focusses on finding failures as quickly as possible and serves as secondary strategy, in case the LNS is not able to improve anymore (Vilím et al., 2015).

5

Problem description

This chapter presents the description of the multi-product maritime inventory routing problem with undedicated compartments. We introduce the concepts that define this problem, such as the structure of the distribution network, products and fleet. Throughout this chapter, we name the assumptions and additional requirements that underlie the case study that we analyze in Chapter 7. In this explanation, we mainly follow the lines of our literature review in Chapter 2, therefore it is possible to place our problem in the context of other problems in the literature. Later, in Chapter 6 we formulate mathematical models for the problem that we introduce here.

The multi-product maritime inventory routing problem concerns the transportation of multiple bulk products between ports, as well as the allocation of products to the compartments on a ship. The objective is to find an optimal delivery schedule that minimizes transportation costs and penalties for the violation of inventory constraints over a given planning horizon.

Time horizon
The planning horizon T of our problem spans from one up to six months. We discretize the planning horizon into a set of time points $\mathcal{T}$, indexed by t. All time points are at equal distance of each other, given a time granularity of one day.

Network
The ports in the problem can be classified as either production ports (refineries), where there is supply of products, or consumption ports, where there is demand. In the general case, the maritime inventory routing problem may contain multiple refineries and ports that are both production and consumption ports, but we restrict ourselves to the case

where there is exactly one refinery, and where every other port only consumes products. From now on, we address the single production port in our problem as the *refinery*. The set of ports is denoted by $\mathcal{P}$, whereas $\mathcal{P}^C \subseteq \mathcal{P}$ is the set of all consumption ports. We index a port by the letter i or j, while we use r for the single refinery.

Products

The maritime inventory routing problem deals with the transportation of continuous product, more specifically petroleum products. The set of products is $\mathcal{K}$. This set is strictly partitioned into a subset of base products K^B and of end products K^E. Base products are the products that are produced and held at the refinery, whereas end products are consumed and held at consumption ports. When products are loaded into a compartment at the refinery, an instantaneous conversion from base products into end products takes place. Some end products are corresponding to exactly one base product, whereas other end products are a mixture of multiple base products. We call the translation of base products into end products *product conversion*. The conversion function $F : K^B \rightarrow K^E$ is a linear transformation that maps quantities of base products into quantities of end products, i.e., how much of each base product needs to be mixed together to obtain one unit of the corresponding end product.

Fleet

The transportation of products is done using ships. The fleet of ships, denoted by set $\mathcal{S}$, is heterogeneous, meaning that every ship has different characteristics. For each ship s, the travel time between ports i and j is given by T^s_{ij} and depends on the sailing speed of ship s. The travel times are rounded off to the nearest integer in order to fit in the discrete time of the model. Likewise, we can define costs C^s_{ij} associated with travelling between each pair of ports. These costs include the costs of operating at the departure port.

Inventory

Every consumption port $i \in \mathcal{P}^C$ consumes one or more products during the entire planning horizon. For each product $k \in \mathcal{K}^E$ that a port i consumes, it has a separate storage facility with lower and upper inventory limits $\underline{I}_{ik}$ and $\overline{I}_{ik}$. The upper limit corresponds to the storage capacity at a port, whereas the lower limit is a possible required safe stock that the port should hold. At the beginning of the planning horizon, the initial inventory I^0_{ik} for each product at each port is known.

In a similar fashion, we define I^0_{rk}, $\underline{I}_{rk}$, $\overline{I}_{rk}$ and R^P_{rkt} as the initial inventory, inventory lower bound, inventory upper bound and rate of production for refinery r respectively. These parameters are defined only for base products $k \in \mathcal{K}^B$.

In order to deal with excess of products at the refinery or shortage of products at the consumption ports, we impose a back-order policy. For consumption ports, this means

that consumption that cannot be met at a certain time point, needs to be served at a later point in time, with a possible delay. We impose a penalty α_{ikt} for the amount of product that the level of backorders at every time point $t \in \mathcal{T}$. In case of excess products at the refinery, additional storage space needs to arranged at a cost α_{rkt}, resulting in a penalty term in the objective function of the model. The implementation of this inventory policy is explained in more detail in the model formulations for MIP and CP in Chapter 6.

Production and consumption rates
The daily consumption rate R^C_{ikt} is varying and given for each time step t in the entire planning horizon. Note that these parameters for consumption ports are only defined for end products $k \in \mathcal{K}^E$. Similarly, we define R^P_{rkt} as the rate of production for base product k at refinery r respectively. These parameters are only defined for base products $k \in \mathcal{K}^B$.

Time
One important decision that needs to be made, is how to represent time in our models. As outlined in Section 2.2.1, there exist models for our problem in the literature that rely on either a discrete or a continuous representation of time. In this thesis, we develop a model that is based on discrete time. The reason for this is twofold: firstly, discrete-time and continuous-time models handle production and consumption rates in a different way. As pointed out by Christiansen, Fagerholt, Nygreen et al. (2013), when dealing with varying production and consumption rates, it is widely accepted to use discrete time, as no simplifying assumptions need to be made on the production and consumption rates of products. Another reason to use discrete time, is that constraint programming models use a discrete variable to represent time. If we want to compare MIP to CP, the use of discrete time allows for a fair comparison between the two models.

Scheduling
We will address the scheduling mechanisms for both the MIP and the CP model in more detail in the following sections. Next to that, there are some requirements with regard to the scheduling of ships:

- Voyages: As part of a ship schedule, we define a voyage as a trip that starts with a refinery visit and ends at the subsequent refinery visit. One voyage thus contains exactly one refinery visit and all intermediate visits at other ports. A ship schedule can consist of one or multiple voyages. In the constraint programming model, we will leverage the concept of voyages in the model formulation, in order to reduce the solution space for the model. This will be explained in more detail in Section 6.3.

- Weight restrictions: Some ports have weight restrictions based on the maximal allowed draught that a ship can have when leaving the refinery or entering a consump-

tion port. For port i and ship s, this maximum allowed weight is W_i^s.

- Ship unavailability: It might be possible that a ship is unavailable for a certain period of time, due to for example maintenance. This means that the ship returns to the refinery and stays there during this unavailability period. The set periods of unavailability of ship s is $\mathcal{U}^s$.

- Operation times: The duration of operating at a port is assumed to be fixed to a constant time O_i^s. This duration may vary for each ship-port pair, but does not depend on the quantity that is (un)loaded.

- Unloading quantity: When operating at a port, there is both a minimum and maximum quantity that can be unloaded. The lower bound is denoted by $\underline{Q}_{ik}$. The upper bound is equal to the minimum of the compartment capacity $\overline{L}_{ck}^s$ and port capacity $\overline{I}_{ik}$.

Compartments

Each ship s has a given number of compartments, which are not dedicated to a certain product. The set of all compartments of a ship s is $\mathcal{C}^s$. At most one product can be transported in a compartment c at the same time. A ship can carry multiple products in different compartments during the same transport. Each compartment has a variable maximum load capacity $\overline{L}_{ck}^s$, which depends on the density of the product that is loaded into the compartment. When the ship is at a consumption port, it can unload the products that are present in each compartment. It is allowed to have partial (un)loading, meaning that a compartment does not need to be filled or emptied completely. At the beginning of the planning horizon, we assume that the ships have empty compartments and that they are located at the refinery.

6

Model description

In this chapter, we give the formulations of two different mathematical models for the maritime inventory routing problem that was described in Chapter 5. The first model is a mixed integer programming model, whereas the second model is a constraint programming (CP) model.

We organize this chapter as follows: Firstly, we summarize the contributions of the models in this thesis in Section 6.1. Then, in Section 6.2 we formulate the MIP model. In Section 6.3, a model that uses the CP formalism, as introduced in Section 4.2, is given. Lastly, in Section 6.4 we compare the two models.

6.1. Contribution

The models that are presented in this chapter contain elements of models that are available in the literature and new insights. As the MIP formulation for this problem is widely documented, the MIP model, that is described in Section 6.2, is similar to other models in literature. Especially the arc-flow structure of the scheduling part of the model is a core element in most mixed integer programming models that we encountered. The construction of the network is a slight adaptation from common formulation, in order to meet requirements from the case study. Several extensions were made to account for multiple products and undedicated compartments. These adjustments are similar to the adjustments made by Agra, Christiansen et al. (2013) and Foss et al. (2016).

For the constraint programming formulation, much less reference literature was available. To the best of our knowledge, only two articles on constraint programming for the

Table 6.1: This table contains all sets, and constants that correspond to the concepts that are introduced in the problem description in Chapter 5. We repeat these symbols here, before we give the model description.

Sets	
$\mathcal{T} = \{1, \ldots, T\}$	Set of time points t up to planning horizon T
$\mathcal{P}$	Set of all ports i, where $r \in \mathcal{I}$ is the single refinery
$\mathcal{P}^C \subseteq \mathcal{P}$	Set of consumption ports i
$\mathcal{K}$	Set of all products k
$\mathcal{K}^B \subseteq \mathcal{K}$	Set of base products k
$\mathcal{K}^E \subseteq \mathcal{K}$	Set of end products k
$\mathcal{S}$	Set of all ships s
$\mathcal{C}^s$	Set of compartments c of ship s
$\mathcal{U}^s$	Set of unavailability periods u of ship s
Constants	
T	Fixed planning horizon
P	Total number of consumption ports, i.e., $\lvert\mathcal{P}^C\rvert = P$
S	Total number of ships, i.e., $\lvert\mathcal{S}\rvert = S$
I_{ik0}	Initial inventory of product k at port i
$\overline{I}_{ik}$	Maximum inventory capacity of product k at port i
$\underline{I}_{ik}$	Inventory lower bound of product k at port i
R^P_{rk}	Variable production rate of product k at port i
R^C_{ik}	Variable consumption rate of product k at port i
$\underline{Q}_{ik}$	Lower bound on quantity of product k (un)loaded at port i
$\overline{L}^s_{ck}$	Maximum load capacity of compartment c of ship s for end product k
T^s_{ij}	Travel time between ports i and j for ship s
C^s_{ij}	Cost of operating at port i and sailing from port i to port j for ship s
W^s_i	Maximum allowed weight to enter port i for ship s
O^s_i	Operating time for ship i to port s
$F_{k\kappa}$	Conversion factors for one unit of end product κ in terms of units of base products k.

maritime inventory routing problem are published, namely by Goel, Slusky et al. (2015) and Giles et al. (2016), although both articles have a different setting than this thesis. The paper by Goel, Slusky et al. (2015) analyses an inventory routing problem where there is one single product. This product is LNG, which requires different additional side constraints. Giles et al. (2016) solve a supply and delivery problem for a network with suppliers, plants and customers.

In the formulation that is presented in this thesis, elements from the aforementioned articles are combined in order to develop a model that is suitable for our case and also equivalent to the MIP model formulation, such as the use of interval variables to model port visits. For example, mechanisms to regulate the allocation of products to compartments, the use of the pulse function to model the back-order inventory policy and the travel cost objective function are new. Next to that, the use of voyages in the model in order to implicitly reduce the search space of the model is new.

6.2. Mixed integer programming model

In this chapter, we formulate a mixed integer programming model for the maritime inventory routing problem. Our model is based on the model of Foss et al. (2016), which in turn is an extension of the model of Agra, Andersson et al. (2013).

The model that we use is a discrete-time model that is based on an arc-flow formulation. We first describe the construction of an arc-flow network in Section 6.2.1. After that, we formulate the complete model by stating the constraints for routing (Section 6.2.2), (un)loading (Section 6.2.3), product compartment allocation (Section 6.2.4), port inventory levels (Section 6.2.5) and the objective function (Section 6.2.6).

6.2.1. Arc-flow network

As pointed out in Chapter 2, two alternative types of MIP formulations exist for the maritime inventory routing problem: arc-flow and path-flow formulations. As mentioned before, models that are based on a path-flow formulation have poor scaling capabilities, because of the exponential increase in the number of possible routes. Given the size of our real-life case study, and because of its prominent occurrence in the literature, we decided to choose an arc-flow formulation for our model. In this section, we give a mathematical explanation of this network formulation.

In the arc-flow formulation, the problem is regarded as a network model. In fact, for each ship $s \in \mathcal{S}$, we can define a network (or graph) $G^s = (\mathcal{N}^s, \mathcal{A}^s)$. An illustration of graph G^s can be found in Figure 6.1. The set of nodes $\mathcal{N}^s$ consists of nodes (i,t), for each port $i \in \mathcal{P}$ and time period $t \in \mathcal{T}$. Each node (i,t) in the graph represents the possibility to visit the port i during time period t. As we will see later, by adding the correct arcs to this net-

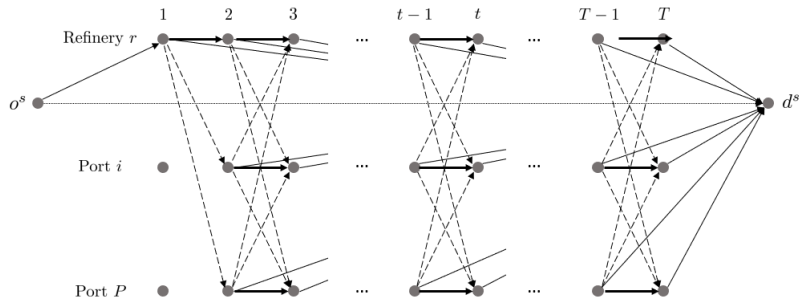

Figure 6.1: Graphical representation of the time-space graph.

work, we can model the movement of a ship as the flow through this network. Next to the nodes (i,t), we add an artifical origin o^s and an artifical destination d^s as two single nodes to the network. These nodes act as respectively a source and sink node in the network. These nodes are not physical ports, but are needed to introduce the entrance and departure of a ship in the network. These nodes correspond to the start and the end of the ship's scheduling in a solution. Therefore,

$$\mathcal{N}^s = (\mathcal{P} \times \mathcal{T}) \cup o^s \cup d^s,$$

where $\times$ is the Cartesian product of two sets.

In order to properly model the movement of a ship through the network, we add directed arcs between selected nodes in the network. These arcs represent possible movements of the ship. We denote an arc a from node (i,t) to (j,τ) as the tuple $(i\,t, j\tau)$. Additionally, we require that a ship enters and leaves the network via the *refinery* r. This means that the first and last stop in a ship schedule are at the refinery. We denote the sets of in-arcs of node $n \in \mathcal{N}^s$ as $\mathcal{A}_n^+$, and out-arcs of this node as $\mathcal{A}_n^-$.

At every point in time, a ship is either sailing between ports, operating at a port or waiting at a port. We add different types arcs to the network in order to model these activities. Because we assume constant travel times, we can combine the activities of operating and sailing to the next port in one type of arc. In fact, we add arcs $a \in \mathcal{A}^s$ of five categories, which are the following:

1. (o^s, d^s): This arc from the artificial origin directly to the artificial destination node represents the possibility of not using a ship at all. This arc needs to be included in order to properly define all possible schedules (with an empty schedule being a trivial possibility), but in practice, this arc will not often be used in an optimal schedule.

2. $(o^s, r1)$: This arc from the artificial origin to the first node of refinery r represents the entering of a ship in the network, starting in the refinery at time t. It might be optimal

for the ships to wait before operating, so a ship can wait at the refinery before starting its first operation, using the arcs of the fourth type.

3. $(r\,t, d^s), \forall\, t \in \mathcal{T}$: These arcs correspond to the end of each ship schedule at any time t. Note that we only include arcs to the artificial destination node from refinery r, which means that each ship must return to the refinery within the planning horizon, in order to terminate its schedule.

4. $(i\,t, i(t+1)), \forall\, i \in \mathcal{P}, t \in \mathcal{T} \setminus T$: These arcs between two successive nodes of the same port represent the possibility to wait at a port i at time t.

5. $(i\,t, j\tau)$, $\forall i, j \in \mathcal{P}, t \in \mathcal{T} \setminus T$, such that $i \neq j$ and $\tau = t + O_i^s + T_{ij}^s$: These arcs model the travel between two ports i and j, taking into account the travel time between the two ports. As mentioned before, we combine operating at port i and sailing to another port j in one arc. We assume that a ship immediately starts sailing to the next port in its schedule upon completion of an operation at a port. Furthermore, when a ship starts its operation in port i at time t before sailing to port j, this ship will arrive at port j at time $\tau = t + O_i^s + T_{ij}^s$, where O_i^s is the operating time of ship s at port i, and T_{ij}^s is the travel time of ship s between ports i and j. We therefore add these arcs to the network.

Note that in the construction of this network, some arcs are inherently infeasible. For example, at the first time step, we require each ship to enter the network at the refinery. This means that it cannot visit other ports at this time. Therefore, these arcs coming from these nodes (and other infeasible arcs) are not included in the network.

6.2.2. Routing

As mentioned earlier, we use the arcs to model the flow of a ship through the network. For each arc that is included in the network, we can define a decision variable that indicates whether this arc is included in a solution to the problem. That is, for each arc $a \in \mathcal{A}^s$, we define a decision variable x_a^s that is equal to 1 if the arc is used, and 0 otherwise. Because every arc a between a port i and destination port j is uniquely characterized by its starting time t, we can use the notation x_{ijt}^s to indicate these decision variables.

Following the definition of the arcs, that a ship immediately leaves a port after completing its operation, the following is true: Binary variable x_{ijt}^s is equal to 1 if ship s is scheduled to start operating at port i at time t and to sail to port $j \neq i$ immediately after the operation period of O_i^s, and 0 otherwise. When $j = i$, x_{iit}^s is equal to 1 if ship s is present in port i and stays there for (at least) one more time period, and 0 otherwise. For $i = o^s$ or $j = d^s$, the arcs represent entering or leaving the network. Using these definitions, we can declare the

constraints that model the routing of ships during time horizon T. The routing constraints can now be defined as follows:

$$x^s_{o^s r1} + x^s_{o^s d^s} = 1, \quad \forall s \in \mathcal{S}, \tag{6.1}$$

$$\sum_{t \in \mathcal{T}} x^s_{r d^s t} + x^s_{o^s, d^s} = 1, \quad \forall s \in \mathcal{S}, \tag{6.2}$$

$$\sum_{a \in \mathcal{A}^+_n} x^s_a = \sum_{a \in \mathcal{A}^-_n} x^s_a, \quad \forall s \in \mathcal{S}, n \in \mathcal{N}^s \setminus (o^s \cup d^s), \tag{6.3}$$

$$x^s_{rrt} = 1, \quad \forall s \in \mathcal{S}, t \in [s_u, e_u), u \in U^s, \tag{6.4}$$

$$x^s_{ijt} \in \{0,1\}, \quad \forall (it, j\tau) \in \mathcal{A}^s, s \in \mathcal{S}. \tag{6.5}$$

The equalities in (A.2) and (6.2) ensure that exactly one route is scheduled for each ship $s \in \mathcal{S}$. This route starts with the arc from the origin port to the refinery and ends with an arc from the refinery to the artificial destination port. In case that the route from the origin directly to the destination port is scheduled, the ship is not used. Constraints (6.3) represents the flow conservation at each node, implying that a ship can only leave a node when it has entered this node. Because our model is built on a directed and acyclic graph, it is not needed to define subtour-elimination constraints. Constraints (6.4) require that during every unavailability period $u = [s_u, e_u)$ of ship s, the ship is forced to be in the refinery. Constraints (6.5) are the binary restrictions for decision variables x^s_{ijt}.

6.2.3. Loading and unloading

The constraints in the previous section model the flow of a ship through a network. In order to model the problem of our interest, we need variables and constraints that deal with the operations of a ship: loading products into its compartments at the refinery and later unloading these products at consumption ports. We will again use network graph G^s in order to define these variables.

For each $s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s$ and $k \in \mathcal{K}^E$ we introduce continuous variable l^s_{tck}. This variable is the load of end product k in compartment c of ship s at the end of time period t. In order to change the load of a product on a ship, we need binary variables o^s_{it}. These are equal to 1 when ship s has an operation at port i that end at time t.

When operating at the refinery, a ship loads products from the refinery inventory into its compartment. We introduce continuous variable q^s_{rtck} for each $s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s$ and $k \in \mathcal{K}^E$, which is the quantity of product k that is loaded into compartment c of ship s during the operation that ends at time t. For consumption ports $i \in \mathcal{P}^C$ the variables q^s_{itck} are defined likewise, except that the product is unloaded from the ship to the port inventory.

Using these variables, we can define the corresponding (un)loading constraints as follows:

$$l^s_{tck} = l^s_{(t-1)ck} + q^s_{rtck} - \sum_{i \in \mathcal{P}^C} q^s_{itck}, \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.6}$$

$$o^s_{it} = \sum_{j \in \mathcal{P}} x^s_{ij(t-O^s_i)}, \quad \forall s \in \mathcal{S}, i \in \mathcal{P}, t \in \mathcal{T}, \tag{6.7}$$

$$q^s_{itck} \geq \underline{Q}_{ik}\, o^s_{it}, \quad \forall s \in \mathcal{S}, i \in \mathcal{P}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.8}$$

$$q^s_{itck} \leq M\, o^s_{it}, \quad \forall s \in \mathcal{S}, i \in \mathcal{P}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.9}$$

$$l^s_{0ck} = 0, \quad \forall s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.10}$$

$$0 \leq l^s_{tck} \leq \overline{L}^s_{ck}, \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.11}$$

$$q^s_{itck} \geq 0, \quad \forall s \in \mathcal{S}, i \in \mathcal{P}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.12}$$

$$o^s_{it} \in \{0,1\}, \quad \forall s \in \mathcal{S}, i \in \mathcal{P}, t \in \mathcal{T}. \tag{6.13}$$

Constraints (6.6) model the change of the load of end product k in compartment c of ship s at time t over time. This load is equal to the load at time $t-1$ plus the quantity that is loaded into c at the refinery minus the quantity that is unloaded from c at consumption ports. In Constraints (6.7) we link the binary operating variables o^s_{it} to the routing decision variables. These constraints state that o^s_{it} must be equal to 1 if ship s departs from port i at time t.

The fact that (un)loading can only take place when a ship is operating at the port is declared in the big-M Constraints (6.8) and (6.9), together with corresponding lower and upper bounds on the quantity that is (un)loaded. In Constraints (6.9), the constant M takes the value that is the minimum of compartment capacity L^s_{ck} and port capacity $\overline{I}_{ik}$, as the quantity cannot be larger than the compartment size or port capacity. Note that these big-M constraints might allow the binary decision variables o^s_{it} to take a value close to zero in the linear relaxation. Therefore, there is a risk that these constraints worsen the linear relaxation of the problem. This problem could be solved for example by adding valid inequalities.

Constraints (6.10) state that all compartments are empty at the beginning of the planning horizon. We declare load variables l^s_{tck} as continuous variables with hard lower and upper bounds in (6.11). Constraints (6.12) declare the non-negative continuous variables q^s_{itck}, whereas Constraints (6.13) are the binary restrictions on o^s_{it}.

6.2.4. Product compartment allocation

In the previous section, we have declared the variables and constraints that model the (un)loading of products into or from ships. One key assumption in our problem is, that all compartments are undedicated to specific products, which means that all products can

be loaded into each compartment.

As our products are non-mixable, we need some constraints in order to ensure that a compartment of a ship can only contain one product at a time. For this, we introduce a binary variable y^s_{tck} for all $s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E$. This variable is equal to 1 when product k is loaded into compartment c of ship s at time t and zero otherwise. The allocation constraints can then be defined as follows:

$$l^s_{tck} \leq \overline{L}^s_{ck} y^s_{tck}, \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.14}$$

$$\sum_{k \in \mathcal{K}^E} y^s_{tck} \leq 1 \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{6.15}$$

$$y^s_{tck} \in \{0,1\}, \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E. \tag{6.16}$$

Constraints (6.14) ensure that whenever a compartment c of ship s contains some product k, the corresponding binary y^s_{tck} should be equal to 1. Constraints (6.15) declare that there can only one product at a time loaded into a compartment. Constraints (6.16) declare binary allocation variables y^s_{tck}.

6.2.5. Port inventory levels

In the previous sections, we have seen how the routing of ships is handled in the mathematical program, together with the variables that account for the load of products in ships. As one of the main goals of this problem is to keep all inventory levels of products between the corresponding bounds, we still need to define a mechanism to linking the above variables to the stock inventory levels.

We formulate the inventory balance constraints for all ports $i \in \mathcal{P}$ and all products $k \in \mathcal{K}$. As the inventory levels of ports, we introduce continuous variables $s_{ikt}, \forall i \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T} \cup 0$. The value of this variable is the inventory level of product k at port i at the end of time interval t. We need to make a distinction between the refinery and consumption ports, as they hold different types of products. Therefore, we only use the variables corresponding to base products for the refinery and the variables corresponding to end products for consumption ports.

As described in Chapter 5, we impose the inventory bounds for some part as a soft constraints. We will describe the penalty variables for the refinery and for consumption ports separately.

For the refinery r, we define a continuous penalty variable for the excess stock of base product $k \in \mathcal{K}^B$ at time t: z_{rkt}. As we mentioned before, we allow the stock level s_{rkt} to be above the maximum inventory bound $\overline{I}_{rk}$ and penalize the difference between the stock

level and the inventory bound, i.e.:

$$z_{rkt} = \max\{s_{rkt} - \overline{I}_{rk}, 0\}. \tag{6.17}$$

Likewise, we can define continuous penalty variable for the shortage of stock of end product $k \in \mathcal{K}^E$ at port i at time t: z_{rkt}. This penalty comes in the form of a backorder, meaning that every unmet demand must be satisfied at a later time. We allow the stock level s_{ikt} to be below the minimum inventory bound $\underline{I}_{rk}$ and penalize the difference between the stock level and the inventory bound, i.e.:

$$z_{ikt} = \max\{\underline{I}_{ik} - s_{ikt}, 0\}. \tag{6.18}$$

The inventory balances are modelled using the following constraints:

$$s_{rkt} = s_{rk(t-1)} - \sum_{s\in\mathcal{S},\, c\in\mathcal{C}^s,\, \kappa\in\mathcal{K}^E} F_{k\kappa} q^s_{rtc\kappa} + R^P_{rkt}, \quad \forall k \in \mathcal{K}^B, t \in \mathcal{T}, \tag{6.19}$$

$$s_{ikt} = s_{ik(t-1)} + \sum_{s\in\mathcal{S},\, c\in\mathcal{C}^s} q^s_{itck} - R^C_{ikt}, \quad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}, \tag{6.20}$$

$$z_{rkt} \geq s_{rkt} - \overline{I}_{rk}, \quad \forall k \in \mathcal{K}^B, t \in \mathcal{T}, \tag{6.21}$$

$$z_{ikt} \geq \underline{I}_{ik} - s_{ikt}, \quad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}, \tag{6.22}$$

$$s_{ik0} = I_{ik0}, \quad \forall i \in \mathcal{P}, k \in \mathcal{K}, \tag{6.23}$$

$$s_{rkt} \geq 0, \quad \forall k \in \mathcal{K}^B, t \in \mathcal{T}, \tag{6.24}$$

$$s_{ikt} \leq \overline{I}_{ik}, \quad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}, \tag{6.25}$$

$$z_{ikt} \geq 0, \quad \forall i \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T}. \tag{6.26}$$

Constraints (6.19) are the inventory balance constraints for all base products k at refinery r at time steps t. The constraints state that the inventory level at time t is equal to the inventory level at time $t-1$, minus the amount that is loaded into ships, plus the amount of refinery production R^P_{rkt} at time t. The factor $F_{k\kappa}$ accounts for the conversion between base products k at the port and the end products κ that are loaded into the ships. Similar inventory balances for consumption ports are given in Constraints (6.20). Note that at consumption ports, no conversion is needed, because only end products are consumed and delivered here.

Constraints (6.21) are equivalent definitions of the penalty variables for the refinery stock levels, given in (6.17).Together with the non-negativity requirements in (6.26), these constraints form a linear reformulation of the original definition. In a similar way, (6.22) and (6.26) form a linear reformulation of Constraints (6.18). Because all variables z_{ikt} are minimized in the objective function (see Section 6.2.6), the expressions for z_{ikt} are equal

to the excess stock or shortage during the process of optimization.

Constraints (6.23) fix the initial inventory levels to I_{ik0}. In Constraints (6.24) and (6.25) we declare the non-negative continuous inventory level variables s_{ikt} together with the hard bounds on these variables. In order to maintain feasibility at all times, this hard bound is a lower bound for the refinery and an upper bound for consumption ports. Constraints (6.26) declares non-negative continuous penalty variables.

6.2.6. Objective function

The objective of our problem is to minimize both the total (travel and operational) cost C^s_{ij}, which is equal to the sum over all arcs of the travel costs, and sum of all penalties z_{ikt} for shortage or excess of all products at all ports and all time steps. This penalty cost is weighted with a time-dependent weighting factor α_{ikt}. Therefore, we can formulate the objective function as follows:

$$\min \sum_{s\in\mathcal{S},\,(it,j\tau)\in\mathcal{A}^s} C^s_{ij} x^s_{(it,j\tau)} + \sum_{i\in\mathcal{P},\,k\in\mathcal{K},\,t\in\mathcal{T}} \alpha_{ikt} z_{ikt}. \tag{6.27}$$

Table 6.2: This table contains all variables that are used in the MIP model.

Variables	
x^s_{ijt}	Binary variable that is 1 if ship $s \in \mathcal{S}$ uses arc $(it, j\tau)$ in the arc-flow network. This arc can be one of 5 categories described in Section 6.2.1
l^s_{tck}	Continuous variable for the quantity of product $k \in \mathcal{K}^E$ in compartment $c \in \mathcal{C}^s$ of ship $s \in \mathcal{S}$ at time $t \in \mathcal{T}$
q^s_{rtck}	Continuous variable for the quantity of end product $k \in \mathcal{K}^E$ loaded to compartment $c \in \mathcal{C}^s$ of ship $s \in \mathcal{S}$ at refinery r at time $t \in \mathcal{T}$
q^s_{itck}	Continuous variable for the quantity of end product $k \in \mathcal{K}$ unloaded from compartment $c \in \mathcal{C}^s$ of ship $v \in \mathcal{S}$ at consumption port $i \in \mathcal{P}^C$ at time $t \in \mathcal{T}$
o^s_{it}	Binary variable that is 1 if ship $s \in \mathcal{S}$ has started an operation at port $i \in \mathcal{P}$ that is ending at time $t \in \mathcal{T}$
y^s_{tck}	Binary variable that is 1 if end product $k \in \mathcal{K}^E$ is loaded into compartment $c \in \mathcal{C}^s$ of ship $s \in \mathcal{S}$ at time $t \in \mathcal{T}$
s_{ikt}	Continuous variable for the inventory level of product $k \in \mathcal{K}$ at port $i \in \mathcal{P}$ at time $t \in \mathcal{T}$
z_{rkt}	Continuous variable for the excess inventory level of base product $k \in \mathcal{K}^B$ at refinery r at time $t \in \mathcal{T}$
z_{ikt}	Continuous variable for the shortage of end product $k \in \mathcal{K}^E$ at consumption port $i \in \mathcal{P}^C$ at time $t \in \mathcal{T}$

6.3. Constraint programming model

In Section 6.2, we formulated a mixed integer programming model for the maritime inventory routing problem. In this section, we formulate a model that is based on the concepts of constraint programming for scheduling problems that we introduced in Section 4.2. The model is based on what is done in earlier work on constraint programming for maritime inventory routing by Goel, Slusky et al. (2015) and Giles et al. (2016), but is modified in order to incorporate specific demands from our case study. Next to that, we introduce the concept of voyages in order to reduce possible solution space of the model.

The CP model is built up in the same way as the mixed integer programming model. We first introduce the interval variables that correspond to the port visits in Section 6.3.1. In Sections 6.3.2-6.3.6 we explain the parts of the model that regulate routing, (un)loading, product compartment allocation, port inventory levels and objective function in that order.

6.3.1. Port visits and voyages

As described in Section 4.2, we can model a scheduling problem using optional interval variables. In our model, the visits that ships make at each port are the interval variables that can be scheduled. In order to properly define the interval variables, we introduce constant J^s as the predefined number of voyages a ship s at most can make. The corresponding set of possible voyages of ship s is $\mathcal{J}^s = \{1, \ldots, J^s\}$. We define the port visits via the optional interval variables x^s_{ij} for $s \in \mathcal{S}, i \in \mathcal{P}, j \in \mathcal{J}^s$. These variables represent the stop at port i during the j-th voyage of ship s. These optional interval variables have an integer length of at least 1 and, if present, their domain is equal to all possible half-open intervals with discrete start and end times within the planning horizon T, i.e.,

$$\operatorname{dom} x^s_{ij} = \{\perp\} \cup \{\,[s,e) \mid s,e \in \mathbb{Z},\ s,e \in [0,T],\ e \geq s+1\}\,.$$

Next to that, we define the *voyage* variables x^s_j as the j-th voyage of ship s, for $s \in \mathcal{S}, j \in \mathcal{J}^s$. Remember that we defined a voyage as a trip that starts with a stop at refinery r and ends at the subsequent refinery stop. Intuitively, we can see voyage variable x^s_j as the collection of all possible port stops that a ship s can make during its j-th voyage. We formalize this definition in the constraints below. The domain of the voyage variables is equal to that of the port stop variables, i.e.,

$$\operatorname{dom} x^s_j = \{\perp\} \cup \{\,[s,e) \mid s,e \in \mathbb{Z},\ s,e \in [0,T],\ e \geq s+1\}\,.$$

In order to define the relation between variables x^s_j and x^s_{ij}, remember the definition of the `span` constraint (Section 4.2). We use this constraint as follows:

$$\texttt{span}\left(x_j^s, \left\{x_{ij}^s\right\}_{i\in\mathcal{P}}\right), \quad \forall j \in \mathcal{J}^s, s \in \mathcal{S}. \tag{6.28}$$

According to the definition of span, this states the following: x_j^s is present if and only if at least one of $\left\{x_{ij}^s\right\}_{i\in\mathcal{P}}$ is present, and the start of x_j^s coincides with the earliest start of these intervals.

In this way, Constraints (6.28) ensure that the voyage variables cover all corresponding port stop variables. However, in order to complete satisfy the definition of a voyage, namely that a voyage starts with a refinery stop and contains at least one stop at another port, we pose the following constraints:

$$\mathit{presenceOf}\left(x_j^s\right) \quad \Rightarrow \quad \mathit{presenceOf}\left(x_{rj}^s\right), \quad \forall j \in \mathcal{J}^s, s \in \mathcal{S}, \tag{6.29}$$

$$\mathit{presenceOf}\left(x_j^s\right) \quad \Rightarrow \quad \bigvee_{i\in\mathcal{P}^C} \mathit{presenceOf}\left(x_{ij}^s\right), \quad \forall j \in \mathcal{J}^s, s \in \mathcal{S}, \tag{6.30}$$

$$\texttt{startAtStart}\left(x_j^s, x_{rj}^s\right), \quad \forall j \in \mathcal{J}^s, s \in \mathcal{S}. \tag{6.31}$$

Constraints (6.29), (6.30) and (6.31) together ensure that the interval variables for voyages model the definition correctly. Constraints (6.29) and (6.30) declare that when a ship s makes its j-th voyage, this voyage must include its j-th visit to the refinery r and at least one corresponding stop at one of the consumption ports. Constraints (6.31) define that a voyage starts with a visit to the refinery r before travelling to other ports.

We end this section by posing constraints that eliminate symmetries among voyage variables and declare the variables x_j^s and x_{ij}^s:

$$\mathit{presenceOf}\left(x_j^s\right) \Rightarrow \mathit{presenceOf}\left(x_{(j-1)}^s\right), \quad \forall j \in \mathcal{J}^s \setminus \{1\}, s \in \mathcal{S}, \tag{6.32}$$

$$\texttt{endBeforeStart}\left(x_{(j-1)}^s, x_j^s\right), \quad \forall j \in \mathcal{J}^s \setminus \{1\}, s \in \mathcal{S}, \tag{6.33}$$

$$x_j^s \in \{\perp\} \cup \{[s,e) \mid s,e \in \mathbb{Z},\ s,e \in [0,T],\ e \geq s+1\}, \quad \forall j \in \mathcal{J}^s, s \in \mathcal{S}, \tag{6.34}$$

$$x_{ij}^s \in \{\perp\} \cup \{[s,e) \mid s,e \in \mathbb{Z},\ s,e \in [0,T],\ e \geq s+1\}, \quad \forall i \in \mathcal{P}, j \in \mathcal{J}^s, s \in \mathcal{S}. \tag{6.35}$$

Constraints (6.32) and (6.33) eliminate symmetries among voyages. This means that voyages are included in a solution in ascending time order, because whenever the j-th voyage of a ship is present, Constraints (6.32) require that the voyage before is also present. The time ordering of port visits is implied by Constraints (6.33). Constraints (6.34) and (6.35) declare the optional interval variables within their domain.

6.3.2. Routing

In order to control the scheduling of all port visits and voyages of a ship s, we use the sequence variables that we introduced in Section 4.2. In order to properly define these variables, we consider the set of all its possible port stops $\mathcal{X}^s$. This set consists of the following elements:

- All possible port stops x^s_{ij}, as defined in the previous section.
- An artificial first stop x^s_0. This stop is related to the arc from the artifical origin node in the MIP model. We need this stop in order to ensure that a ship starts its schedule at the refinery, as explained in Chapter 6.
- Similarly, we define an artificial last stop x^s_T, that we need to ensure that a ship terminates its schedule at the refinery.
- Unavailability periods x^s_u from set U^s.

We define $\mathcal{X}^s$ as the union of all elements that we listed above:

$$\mathcal{X}^s = \left\{ \left\{ x^s_{ij} \right\}_{i \in \mathcal{P}, j \in \mathcal{J}^s} \cup x^s_0 \cup x^s_T \cup \left\{ x^s_u \right\}_{u \in U^s} \right\}.$$

Additionally, we can define type functions $\Theta^s : \mathcal{X}^s \to \mathcal{P}$, that for each port stop in $\mathcal{X}^s$ returns the port where this stop takes place, i.e.:

$$\Theta^s(x) = \begin{cases} i & \text{if } x = x^s_{ij}, \\ r & \text{if } x = x^s_0, \text{ if } x = x^s_T, \text{ or if } x = x^s_u. \end{cases}$$

The intuition behind this definition is as follows: for any ordinary port stop x^s_{ij}, the type function returns port i. The type of the first and last port visits are equal to refinery r. Next to that, we require a ship to be at the refinery during unavailability periods x^s_u, hence $\Theta^s(x^s_u) = r$.

Now that we have defined sets $\mathcal{X}^s$ and functions Θ^s, we can introduce sequence variables over all possible port stops of a ship s. This declaration is also part of our model:

$$\pi^s = \mathit{sequenceVar}\left(\mathcal{X}^s, \Theta^s\right). \tag{6.36}$$

Constraints (6.36) define the sequence variables π^s over the set of all possible port stops $\mathcal{X}^s$ of a ship s. We can use this sequence variable to define the routing constraints of a ship s as follows:

$$\texttt{first}(\pi^s, x_0^s), \quad \forall s \in \mathcal{S}, \tag{6.37}$$

$$\texttt{last}(\pi^s, x_T^s), \quad \forall s \in \mathcal{S}, \tag{6.38}$$

$$\texttt{noOverlap}(\pi^s, T^s), \quad \forall s \in \mathcal{S}, \tag{6.39}$$

$$x_0^s = [0, 1), \quad \forall s \in \mathcal{S}, \tag{6.40}$$

$$x_T^s \subseteq [1, T), \quad \forall s \in \mathcal{S}, \tag{6.41}$$

$$x_u^s = [s_u, e_u), \quad \forall s \in \mathcal{S}, u \in \mathcal{U}^s. \tag{6.42}$$

Constraints (6.37) and (6.38) state that the first and last stop in sequence variable π^s are x_0^s and x_T^s, respectively. Because we defined $\Theta(x_0^s) = \Theta(x_T^s) = r$, this is equivalent to requiring that the schedule for ship s starts and ends at the refinery. Then, the `noOverlap` constraints in (6.39) state that no two stops in π^s can overlap. Furthermore, the time between two stops at ports i and j in π^s should be at least T_{ij}^s. This information is added through transition matrix T^s. Additionally, these constraint ensure that no other stop can be scheduled during a scheduled unavailability x_u^s, which is also included in π^s.

Constraints (6.40) declare the first visits of all ships s at the beginning of the planning horizon, whereas Constraints (6.41) declare the last visits, which could take place during any time. Constraints (6.42) declare the unavailability periods for ship s as fixed interval variables.

6.3.3. Loading and unloading

Now that we have defined the routing mechanisms of the constraint programming model, we shift the focus to the modelling of (un)loading operations. For this, we make use of cumulative functions, which are described in Section 4.2.1. Remember the two types of elementary cumulative functions:

- *pulse*$(x, h_{\min}, h_{\max})$: a function that has a variable height between $h_{\min}$ and $h_{\max}$ over the length of optional interval variable x and 0 elsewhere. If x is absent, the `pulse` function has height 0 everywhere.

- *stepAtEnd*$(x, h_{\min}, h_{\max})$: a function that has a variable height between $h_{\min}$ and $h_{\max}$ after the end of optional interval variable x and 0 elsewhere. If x is absent, the *stepAtEnd* function has height 0 everywhere.

Cumulative functions are defined as sums of elementary cumulative functions, i.e. they are piecewise linear functions. We denote step, pulse and cumulative functions with a ~ on top of the variable.

We introduce integer (un)loading variables, analogous to variables in the MIP model that we formulated in Section 6.2. The non-negative step function $\tilde{q}^s_{ijck}$ is the quantity of product $k \in \mathcal{K}^E$ that is loaded into or from compartment c at port i during the j-th voyage of ship s. This function has variable height between minimum (un)loading quantity $\underline{Q}_{ik}$ and maximum compartment capacity $\overline{L}^s_{ck}$. If port stop x^s_{ij} is absent, the height of $\tilde{q}^s_{ijck}$ is equal to 0. We formulate this as follows:

$$\tilde{q}^s_{ijck} = stepAtEnd\left(x^s_{ij}, \underline{Q}_{ik}, \overline{L}^s_{ck}\right), \quad \forall s \in \mathcal{S}, i \in \mathcal{P}, j \in \mathcal{J}^S, c \in \mathcal{C}^S, k \in \mathcal{K}^E. \tag{6.43}$$

We can use the (un)loading variables to define cumulative functions $\tilde{l}^s_{ck}$ that denote the cumulative quantity of end product $k \in \mathcal{K}^E$ that is loaded into compartment c of ship s:

$$\tilde{l}^s_{ck} = \sum_{j \in \mathcal{J}^s} \tilde{q}^s_{rjck} - \sum_{i \in \mathcal{P}^C} \sum_{j \in \mathcal{J}^s} \tilde{q}^s_{ijck}, \quad \forall s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E. \tag{6.44}$$

The load of product k in compartment c of ship s is modelled as cumulative function $\tilde{l}^s_{ck}$ in (6.44). This load is composed of all quantities that are loaded into the compartment at refinery r, minus the quantities that are unloaded at consumption ports. Because of the assumption, that all compartments are empty at the beginning of the planning horizon, we do not need to include initial loads in this expression.

The values that load $\tilde{l}^s_{ck}$ have have hard bounds, as the load cannot be negative and cannot exceed the tank capacity $\overline{L}^s_{ck}$:

$$\texttt{alwaysIn}\left(\tilde{l}^s_{ck}, 0, T, 0, \overline{L}^s_{ck}\right), \quad \forall s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E. \tag{6.45}$$

The hard bounds on the quantity of product that a ship can transport in compartment c are given by the `alwaysIn` constraints in (6.45). Following the definition of `alwaysIn` in Section 4.2, Constraints (6.45) ensure that load variables $\tilde{l}^s_{ck}$ only have values between 0 and $\overline{L}^s_{ck}$ during planning horizon $[0, T]$.

6.3.4. Product compartment allocation

Similar to the MIP model in Section 6.2, we need constraints to ensure that a compartment can contain only one product at a time. Once a certain product is loaded into a compartment at the refinery, this will be the only product type in this compartment during an entire voyage, because there cannot be any products loaded into the compartments at consumption ports. We will make use of this insight and the voyage formulation to model this in a compact way, as we only need one allocation variable per voyage.

With this in mind, we can introduce optional interval variables y^s_{jck}, which are present when product $k \in \mathcal{K}^E$ is loaded into compartment c during the j-th voyage of ship s and

absent otherwise. The allocation constraints are then stated as follows:

$$\texttt{alternative}\left(x^s_j, \left\{y^s_{jck}\right\}_{k\in\mathcal{K}^E}\right), \quad \forall s\in\mathcal{S}, j\in\mathcal{J}^s, c\in\mathcal{C}^s, \tag{6.46}$$

$$\left(q^s_{rjck} > 0\right) \Rightarrow presenceOf\left(y^s_{jck}\right), \quad \forall s\in\mathcal{S}, j\in\mathcal{J}^s, c\in\mathcal{C}^s, k\in\mathcal{K}^E, \tag{6.47}$$

$$y^s_{jck} \quad \subseteq \{\perp\}\cup[0,T), \quad \forall s\in\mathcal{S}, j\in\mathcal{J}^s, c\in\mathcal{C}^s, k\in\mathcal{K}^E. \tag{6.48}$$

The `alternative` constraints in (6.46) ensure that whenever a ship s makes its j-th voyage, exactly one of the product compartment allocation variables is present and coincides with this voyage. This means that exactly one product $k\in\mathcal{K}^E$ can be loaded into compartment $c\in\mathcal{C}^s$. Additionally, the constraints in (6.47) ensure that y^s_{jck} is present whenever the continuous loading variable q^s_{ijck} is greater than zero at the refinery. From (6.46) we conclude that only one product can have a positive quantity loaded into compartment c during a voyage. Remember that Constraints (6.45) from Section 6.3.3 ensure that a product can only be unloaded from a compartment c when the load of this compartment is larger than zero.

Following the reasoning above, Constraints (6.46) and (6.47) together with Constraints (6.45) ensure that only one product can be loaded into a compartment. The product compartment allocation variables are declared in (6.48).

6.3.5. Port inventory levels

In order to complete the model, we need to model the inventory balance of the ports $i\in\mathcal{P}$ for all products $k\in\mathcal{K}$. Similar to the approach in Section 6.2.5, we use variables for the inventory level of product k at port i during the planning horizon. In this case, we introduce cumulative functions $\tilde{s}_{ik}$ for this purpose. We only use variables corresponding to base products for the refinery and variables that corresponds to end products for consumption ports.

First, remember that at each time step t, the quantity of base product k that refinery r produces is R^P_{rk}. Similarly, the quantity of end product k that consumption port i consumes is equal to R^C_{ik}. In our CP model, we model the production and consumption as a step functions:

$$\tilde{R}^P_{rk} = \sum_{t\in\mathcal{T}} stepAt\left(t, R^P_{rk}\right), \quad \forall k\in\mathcal{K}^B, \tag{6.49}$$

$$\tilde{R}^C_{ik} = \sum_{t\in\mathcal{T}} stepAt\left(t, R^C_{ik}\right), \quad \forall i\in\mathcal{P}^C, k\in\mathcal{K}^E. \tag{6.50}$$

In Constraints (6.49) and (6.50) we model the cumulative quantity of product k that is produced or consumed at refinery r and consumption ports i as cumulative functions.

In Section 6.3.3 we defined elementary cumulative functions $\tilde{q}^s_{rjck}$ for the quantity of

end product k that was loaded into compartment c of ship s during the ship's j-th stop at refinery r. Because the refinery only has base products in stock, we would like to derive a similar expression for base products k. If the quantity $\tilde{q}^s_{rjc\kappa}$ of an end product κ is loaded into a ship at the refinery, $F_{k\kappa}\tilde{q}^s_{rjc\kappa}$ is taken from the refinery stock of base product k. Therefore the total amount that is subtracted from the stock of base product k during an operation, is the sum of all contributions $F_{k\kappa}\tilde{q}^s_{rjc\kappa}$ over all end products κ:

$$\tilde{q}^s_{rjck} = \sum_{\kappa \in \mathcal{K}^E} F_{k\kappa}\tilde{q}^s_{rjc\kappa}, \quad \forall k \in \mathcal{K}^B. \tag{6.51}$$

We can sum this over all ships, voyages and compartments to find:

$$\tilde{q}_{rk} = \sum_{s \in \mathcal{S}} \sum_{j \in \mathcal{J}^S} \sum_{c \in \mathcal{C}^S} \sum_{\kappa \in \mathcal{K}^E} F_{k\kappa}\tilde{q}^s_{rjc\kappa}, \quad \forall k \in \mathcal{K}^B. \tag{6.52}$$

We can derive a similar expression for the cumulative quantities of end products k that are unloaded from ships to consumption ports i. In this expression, we do not need conversion factor $F_{k\kappa}$:

$$\tilde{q}_{ik} = \sum_{s \in \mathcal{S}} \sum_{j \in \mathcal{J}^S} \sum_{c \in \mathcal{C}^S} \tilde{q}^s_{ijck}, \quad \forall i \in \mathcal{P}, k \in \mathcal{K}^E. \tag{6.53}$$

Next to that, we need the penalty variable for shortages or excesses of a product k at ports i to model the backorder inventory policy. In order to model this using the CP formalism, we use introduce optional interval variables ζ_{ikt} which are present whenever a penalty occurs at time t. Then the penalty can be modelled using an elementary cumulative function $\tilde{z}_{ikt}$ which takes the height of the stock violation in case a penalty occurs. Integer variables z_{ikt} have the value of the penalty of product k at port i in time t and can be added to the objective function:

$$\tilde{z}_{ikt} = pulse(\zeta_{ikt}, z_{ikt}), \quad \forall i \in \mathcal{P}, k \in \mathcal{K}, \tag{6.54}$$

$$\tilde{z}_{ik} = \sum_{t \in \mathcal{T}} \tilde{z}_{ikt} \quad \forall i \in \mathcal{P}, k \in \mathcal{K}, \tag{6.55}$$

$$\zeta_{ikt} \subseteq \{\perp\} \in [t, t+1), \quad \forall i \in \mathcal{P}, k \in \mathcal{K}^E, t \in \mathcal{T}, \tag{6.56}$$

$$z_{ikt} \in \mathbb{Z}_{\geq 0}, \quad \forall i \in \mathcal{P}, k \in \mathcal{K}^E, t \in \mathcal{T}. \tag{6.57}$$

Constraints (6.54) model the loss penalty for product k that occur at port i at time t. Constraints (6.55) give the cumulative penalty function for product k at port i as the sum over all time periods of $\tilde{z}_{ikt}$. In (6.56) we declare the optional interval variables for losses at time t. Non-negative integer penalty variables z_{ikt} are declared in (6.57).

In the above paragraphs, we explained most elements that have an impact on inventory levels $\tilde{s}_{rk}$ for base product k at refinery r. For completeness, we list all contributions to $\tilde{s}_{rk}$ here:

- Initial inventory S^0_{rk}. We can model this as an elementary cumulative function as follows: $\tilde{S}^0_{rk} = stepAt(0, S^0_{rk})$.
- Production as a cumulative function: $\tilde{R}^P_{rk}$ (see Constraints (6.49))
- Products that are loaded into ships: $\tilde{q}_{rk}$ (see Constraints (6.52)).
- Penalties as a cumulative function: $\tilde{z}_{rk}$ (see Constraints (6.55))

This results in the following expression for inventory levels $\tilde{s}_{rk}$:

$$\tilde{s}_{rk} = \tilde{S}^0_{rk} + \tilde{R}^P_{rk} - \tilde{q}_{rk} - \tilde{z}_{rk}, \quad \forall k \in \mathcal{K}^B. \tag{6.58}$$

We can find an equivalent expression for consumption ports:

$$\tilde{s}_{ik} = \tilde{S}^0_{ik} - \tilde{R}^P_{ik} + \tilde{q}_{ik} + \tilde{z}_{ik}, \quad \forall k \in \mathcal{K}^B. \tag{6.59}$$

We only need to impose inventory bounds on cumulative functions $\tilde{s}_{ik}$. For this, we again use the `alwaysIn` constraint:

$$\texttt{alwaysIn}(\tilde{s}_{ik}, 0, T, \underline{I}_{ik}, \overline{I}_{ik}), \quad \forall i \in \mathcal{P}, k \in \mathcal{K}. \tag{6.60}$$

The `alwaysIn` constraints in (6.60) ensure that $\tilde{s}_{ik}$ can only take values of $\underline{I}_{ik}$ and $\overline{I}_{ik}$. The presence of $\tilde{z}_{ik}$ in (6.58) and (6.59) turn this constraint into a one-sided soft constraint.

6.3.6. Objective function

The complete constraint programming model is given by Constraints (6.28)-(6.60). The objective of this model is the same as in Section 6.2.6, namely to minimize both the total cost TC^s, which is composed of travel and operational cost, and the penalties for shortage or excess of all products at all ports. This penalty cost is weighted with a weighting factor α_{ikt}. Therefore, we can formulate the objective function as follows:

$$\min \sum_{s \in \mathcal{S}} TC^s + \sum_{i \in \mathcal{P}} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} \alpha_{ikt} z_{ikt}, \tag{6.61}$$

$$\text{where} \quad TC^s = \sum_i C^s_{\theta^s(\pi^s(i-1)), \theta(\pi^s(i))}, \quad \forall s \in \mathcal{S}. \tag{6.62}$$

The total cost of the ship sequences π^s is calculated in (A.30). It consists of the sum of all travel costs between consecutive port visits for ship s.

Sets	
$\mathcal{J}_s$	Set of possible voyages for ship $s \in \mathcal{S}$
Variables	
x_j^s	Optional interval variables for the j-th voyage of ship s, for $j \in \mathcal{J}^s$
x_{ij}^s	Optional interval variables for the port visit to port i during the j-th voyage of ship s, for $j \in \mathcal{J}^s$
x_0^s	Interval variables for the artificial first stop of ship $s \in \mathcal{S}$
x_T^s	Interval variables for the artificial last stop of ship $v \in \mathcal{V}$
q_{ijck}^s	Discrete variable of the quantity of product $k \in \mathcal{K}^E$ that is loaded into or from compartment $c \in \mathcal{C}^v$ of ship $s \in \mathcal{S}$ at the visit to port $i \in \mathcal{P}$ during the j-th voyage, for $j \in \mathcal{J}^s$
$\tilde{q}_{ijck}^s$	Step functions that represent the (un)loading of product $k \in \mathcal{K}^E$ into or from compartment $c \in \mathcal{C}^v$ of ship $s \in \mathcal{S}$ at the visit to port $i \in \mathcal{P}$ during the j-th voyage, for $j \in \mathcal{J}^s$
$\tilde{l}_{ck}^s$	Discrete variable of the load of product $k \in \mathcal{K}^E$ in compartment $c \in \mathcal{C}^s$ of ship $s \in \mathcal{S}$
y_{ijck}^s	Optional interval variables that are present when product $k \in \mathcal{K}^E$ is loaded into compartment $c \in \mathcal{C}^s$ of ship $s\mathcal{S}$ at the end of the visit to port $i \in \mathcal{P}$ during the j-th voyage, for $j \in \mathcal{J}^s$ and absent otherwise
$\tilde{R}_{rk}^P$	Cumulative function of the production of base product $k \in \mathcal{K}^B$ at refinery r
$\tilde{R}_{ik}^C$	Cumulative function of the consumption of end product $k \in \mathcal{K}^E$ at port $i \in \mathcal{P}^C$
$\tilde{s}_{ik}$	Cumulative function of the inventory level of product $k \in \mathcal{K}$ at port $i \in \mathcal{P}$ during the planning horizon.
ζ_{ikt}	Optional interval variables that are present whenever the inventory levels of product $k \in \mathcal{K}$ in port $i \in \mathcal{P}$ at time $t \in \mathcal{T}$ violates its soft bound.
z_{ikt}	Discrete variable for the excess inventory level of base product $k \in \mathcal{K}^B$ at the refinery r at time $t \in \mathcal{T}$, or for the shortage of end product $k \in \mathcal{K}^E$ at consumption port $i \in \mathcal{P}^C$ at time $t \in \mathcal{T}$
$\tilde{z}_{ik}$	Cumulative function of all excess inventory of base product $k \in \mathcal{K}^B$ at the refinery r, or cumulative function of all shortages of end product $k \in \mathcal{K}^E$ at consumption port $i \in \mathcal{P}^C$

6.4. Comparison between MIP and CP model

In this chapter, we presented two alternative models using either mixed integer programming and constraint programming. We can identify several differences between the models, which we will discuss in this section.

One of the main differences between the two models is the way that ship schedules are constructed. In the CP model, this is done using optional interval variables that represent the port visits. These port visits are represented by half open intervals with discrete start and end points that can be scheduled during the planning horizon. This contrasts the MIP model, where we only model the visit implicitly by the navigation of the ships through their respective graphs.

Another difference is the necessity of binary operating variables o^s_{it}, which are needed to impose conditional minima on the variables q^s_{itck} in Constraints (6.8). These minima should only be enforced whenever the ship is operating at that port during that time slot, which can be achieved using the binary variables. In the CP models, this conditionality is modelled directly through the optional interval variables. Whenever such an interval variable is absent, all corresponding constraints, for example on its minimum value, do not need to be satisfied, allowing for a concise formulation without binary variables.

The product compartment allocation is handled differently in both models. In order to ensure that only one product is loaded into a compartment at all times, the MIP model uses binary variables y^s_{tck}. Therefore, the number of binary variables scales with the length of the time horizon. In contrast, the CP model replaces these variables by y^s_{jck}, where the time index is replaced by the index j, corresponding to a predefined number of voyages that is included in the model. In most cases, this number is smaller than the time horizon T.

The last main difference between the two models is, that the CP models use integer variables instead of continuous variables for the loading quantities q, port inventory levels s, ship load levels l and penalties z. The main reason for this is that variables in CP problems have discrete domains. In order to make a fair comparison between the two models, we need to show that for every solution of the MIP problem, with non-integer values for continuous variables, we can find an alternative solution with integer values, that has the same or a better objective value.

This follows from the fact that whenever a continuous variable takes a fractional value, it is always possible to find an alternative integer solution with the same objective value. We formalize this intuition in the following propositions. First, in Proposition 1, we take all conversion factors between base and end products equal to 1.

Proposition 1. Let Π be a maritime inventory routing problem formulated as a mixed integer program using constraints (A.2)-(6.26) and with objective function (6.27).

Let $\mathcal{I}$ be an instance of this problem with the following properties:

1. Travel cost C^s_{ij} and penalty coefficients α_{ikt} are non-zero, otherwise the statement is trivial.
2. Ship capacity $\overline{L}^s_{ck}$ is integer for all $s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E$.
3. Consumption port inventory related constants R^C_{ikt}, S^0_{ik} and $\underline{S}_{ik}$ are integer for all $i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}$.
4. Refinery inventory related constants R^P_{rkt}, S^0_{rk} and $\overline{S}_{rk}$ are integer for all $k \in \mathcal{K}^B, t \in \mathcal{T}$,
5. Conversion factors $F_{k\kappa}$ are either 0 or 1 for all $k \in \mathcal{K}^B, \kappa \in \mathcal{K}^E$.

Now suppose that there exists a solution Σ to $(\Pi, \mathcal{I})$, with a non-integer value (at least) for one of the following variables:

- Inventory stock variable s_{ikt} for $i \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T}$,
- Penalty variable z_{ikt} for $i \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T}$,
- Ship load variable l^s_{tck} for $t \in \mathcal{T}, s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E$,
- (Un)loading quantity q^s_{itck} for $s \in \mathcal{S}, i \in \mathcal{P}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E$,

Then, there exists a solution Σ' that has the same or lower objective value than Σ.

Proof. We will proof this statement by construction. Without loss of generality, we can assume that penalty variables z_{ikt} are either equal to 0, or that equality holds in the corresponding Constraint (6.21) or (6.22). In other words, the penalty variables are equal to the excess or shortage of a product at a port.

First, we show that whenever one of the variables s_{ikt}, z_{ikt} or l^s_{tck} has a non-integer value, there must be a variable q^s_{itck} that has a non-integer value. We show this by contradiction:

- Suppose one of the variables s_{ikt} has a non-integer value, but all variables q^s_{itck} are integer. This contradicts inventory balance constraints (6.19) or (6.20) and the assumption that input parameters for the initial inventory S^0_{ik} and production/consumption rates R^P_{rkt}/R^C_{ikt} are integer.
- Suppose one of the variables z_{ikt} has a non-integer value. Because all parameters $\underline{S}_{ik}$ and $\overline{S}_{ik}$ are integer, equality in Constraints (6.21) or (6.22) assures that the corresponding variable s_{ikt} has a non-integer value as well. Via the reasoning above, there is a variable q^s_{itck} that has a non-integer value in this case.

- If one of the variables l^s_{tck} has a non-integer value, we can also conclude that there must be a variable q^s_{itck}. This follows from similar reasoning as above, using the ship load balance (6.6) and the fact that the ships are empty at the beginning of the planning period (6.10).

From the previous argument, we know that in our non-integer solution Σ, at least one of the variables q^s_{itck} must have a non-integer value. Hence, it suffices to show that we can construct an alternative solution Σ' in which all (un)loading variables q^s_{itck} are integer. Given this variable q^s_{itck} and its lowest index t, no other variable s_{ikt}, z_{ikt} or l^s_{tck} can have non-integer values for times $t' < t$, because of the integer input parameters and the balance Constraints (6.6), (6.19) and (6.20).

Consider the non-integer variable $q^s_{i\tau ck}$ in Σ which has the lowest index t, which we call τ. We show that we can construct a solution Σ' with integer $q^s_{i\tau ck}$, that has a lower objective value than Σ. For this $q^s_{i\tau ck}$, consider two cases: i corresponds to the refinery r, or to a consumption port.

First the case that $q^s_{r\tau ck}$ is non-integer, which means that an non-integer quantity of product k is loaded into compartment c of ship s. By (6.6) we know that the corresponding ship load $l^s_{\tau ck}$ is also non-integer, and therefore strictly smaller than its upper bound $\overline{L}^s_{ck}$. Because of Constraints (6.2), we know that ship s must return to the refinery at least once before the end of the planning period.

All compartments of ship s are empty at the trip back to the refinery. This means that there must be at least one other variable q^s_{itck} that is non-integer, for $i \in \mathcal{P}^C$. For this variable, we know that $t > \tau$, but *before* the ship returns to the refinery again. We now construct solution Σ' by rounding $q^s_{r\tau ck}$ up to the nearest integer, and by increasing q^s_{itck} with the same amount. This is allowed because the ship load variable $l^s_{\tau ck}$ is smaller than $\overline{L}^s_{ck}$. Note that the objective value of Σ' is the same as that of Σ or lower. The higher quantity that is shipped reduces possible excess at the refinery, and reduces possible shortage at consumption port i.

For Σ', we could have the following two cases:

1. Σ' is a solution with integer values for all variables q^s_{itck}, and hence also for all variables s_{ikt}, z_{ikt} and l^s_{tck} (by the starting argument). In this case, we are done with the proof.

2. Σ' is a solution where still one of the variables q^s_{itck} is non-integer. In this case, we return to the starting point of our construction, by choosing the non-integer variable q^s_{itck} in Σ' with the lowest index t. We apply this procedure repeatedly.

Now we take a look at the second case, where the non-integer variable $q^s_{i\tau ck}$ in Σ with the lowest index τ corresponds to a consumption port i. By the same arguments as before

we know that ship load $l^s_{\tau ck}$ is also non-integer, and therefore strictly smaller than its upper bound $\overline{L}^s_{ck}$. Ship s must return to the refinery at least once before the end of the planning period. All compartments of ship s are empty at the trip back to the refinery. This means that there must be at least one other variable $q^s_{\eta tck}$ that is non-integer, for another consumption port $\eta \in \mathcal{P}^C$. For this variable, we know that $t > \tau$, but *before* the ship returns to the refinery again. We now construct solution Σ' by rounding $q^s_{i\tau ck}$ up to the nearest integer, and by *decreasing* $q^s_{\eta tck}$ with the same amount. This is allowed because the ship load variable $l^s_{\tau ck}$ is smaller than $\overline{L}^s_{ck}$. Note that the objective value of Σ' is could in this case be higher, the same or lower than that of Σ, depending on non-zero weights α_{ikt}. If the operation above results in a higher objective value, we decrease $q^s_{i\tau ck}$ by 1 and increase $q^s_{\eta tck}$ by one, leading to a solution with an objective value lower than what we had before. Again, we could have the following two cases:

1. Σ' is a solution with integer values for all variables q^s_{itck}, and hence also for all variables s_{ikt}, z_{ikt} and l^s_{tck} (by the starting argument). In this case, we are done with the proof.

2. Σ' is a solution where still one of the variables q^s_{itck} is non-integer. In this case, we return to the starting point of our construction, by choosing the non-integer variable q^s_{itck} in Σ' with the lowest index t. We apply this procedure repeatedly.

By repeating this procedure until all variables are integer, we can construct a solution Σ' that only has discrete values for the variables s_{ikt}, z_{ikt}, l^s_{tck} and q^s_{itck}. Because of the finite time horizon, there is only be a finite number of variables q^s_{itck}, which means that this process always terminates. □

7

Computational study

In Chapter 6, we have seen two mathematical formulations of the maritime inventory routing problem. One of the main goals of this thesis, is to develop approaches based on both MIP and CP to solve the problem, and to compare the outcomes of the approaches in a computational study. In this chapter, we will outline this analysis. Firstly, in Section 7.1, we describe the data that we use as input for this problem. Secondly, in Section 7.2, we describe the generation and characteristics of additional instances, that are based on the original data. Finally, in Section 7.3 we present and discuss the (preliminary) results of the runtime analysis, where we solve both the MIP and CP model and compare the results.

7.1. Data description

As described in Chapter 5, an instance of the maritime inventory routing problem is given by to input data, related to the sets and constants that are listed in Table 5.1. Our analysis is based on a real-life instance from a petroleum company. This data set provides information about the characteristics of the ships, ports and products and supply and demand profiles over a time horizon of approximately four months. Next to the original data set, we created a set of instances. By including this set of instances, we can extend our analysis and check whether our findings extend to different instances

In Table 7.1, the characteristics of the real-life case are presented. Apart from the number of ports $|\mathcal{N}|$, ships $|\mathcal{V}|$, base products $|\mathcal{K}^B|$, and end products $|\mathcal{K}^E|$, we list three additional properties in the table. These properties can be used in order to evaluate our created instances, in order to ensure that these instances have realistic properties in comparison to the real-life case. The characteristics that we use are the following:

- Density ρ: This characteristic is defined as the total number of stocks of end products

that are held by the ports, divided by the total possible number of stocks, which is equal to the number $|\mathcal{K}^E| \times |\mathcal{N}^C|$. This ratio is an indication of the complexity of the problem in relation to the number of ports and products that are included. The higher ρ is, the harder the problem is.

- Stock capacity-to-consumption $S^{\max}/R^C$: This characteristic is the average ratio between the stock capacity of a port to its average daily production. The ratio is an indication of how often ports need to be visited in order to meet the demand at that port.

- Ship capacity-to-consumption $L^{\max}/R^C$: This characteristic is the ratio of the total loading capacities of all compartments of all ships, divided by the total consumption of all ports over the entire planning horizon. This characteristic is an indication about the level at which the ships in the instance are able to meet the total demand of end products at the ports.

Table 7.1: In this table, we list the properties of the problem instance that is based on real-life data.

Instance	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{K}^B\rvert$	$\lvert\mathcal{K}^E\rvert$	ρ	$S^{\max}/R^C$	$L^{\max}/R^C$
Real-life case	13	2	7	9	0.44	84.52	9.74

7.2. Instances

With the information from the case study, it is possible to create similar instances of the problem, that have the same or similar characteristics. The aim of these instances is to formulate the MIP and CP models for all instances and analyse the run times for all instances. We generate instances of different sizes. In Table 7.2 we list all instances that are created, together with the same characteristics as we introduced in Section 7.1.

As indicated in Table 7.2, we can divide our instances into four groups. This subdivision is based on the size of the instances and the way in which they are created:

- **S1 - S3:** These instances are small instances, which have four ports (of which one is the refinery), two ships and three to four different product types. These instances are created by filtering the data of the original instances and selecting the four ports and products that these ports hold. Because the instances are small, the density characteristic is relatively high for these instances. As we will see later, the instances S1, S2 and S3 can be solved to near optimality by a MIP approach for a short time horizon, which makes these instances good benchmarks for other methods.

- **M1 - M5:** The number of ports in the mid-size instances varies from 5 to 10. The instances are created in a similar way as the small instances, namely by taking a subset

Table 7.2: In this table, we list the properties of all created instances that we use in the computational study.

Instance	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{K}^B\rvert$	$\lvert\mathcal{K}^E\rvert$	ρ	$S^{\max}/R^C$	$L^{\max}/R^C$
S1	4	2	4	3	0.78	75.50	8.52
S2	4	2	4	3	0.78	88.47	4.66
S3	4	2	3	3	0.56	95.37	10.84
M1	5	2	5	5	0.60	97.13	13.02
M2	5	2	6	5	0.72	91.88	4.36
M3	6	2	6	5	0.70	101.09	6.20
M4	8	2	6	6	0.60	93.19	3.52
M5	10	2	7	6	0.52	85.45	5.22
L1	13	2	7	9	0.44	84.52	9.74
L2	13	2	7	9	0.44	84.52	9.74
L3	13	2	7	9	0.44	84.52	9.74
L4	13	2	7	9	0.43	85.96	9.38
L5	13	2	7	9	0.44	69.77	8.17
L6	13	2	7	9	0.37	82.82	12.88
L7	13	2	7	9	0.46	87.68	9.68
L8	13	2	7	9	0.43	90.47	10.68
Case study	13	2	7	9	0.44	84.52	9.74

of the ports and products from the real-life case. By looking at instances of increasing sizes, it is possible to look up to what instance size it is possible to find good solutions using exact solver algorithms on the model formulations as proposed in Chapter 6. Note that the ship capacity-to-consumption ratios instances M2-M5 are on the low side, this is due to some restrictions on the number of compartments that ships have in these instances. Perhaps these instances need to be adjusted somewhat, in order to make them more comparable to the real-life case.

- **L1 - L3:** The large instances in our instance set all have the similar size as our real-life case. For the first three instances, the port indices are permuted, while keeping all other input data the same. In this way, the distances between port, and the accompanying costs, change, but aggregate production and consumption numbers stayed the same for all products. Apart from the travel distances, these instances have the same exactly the same characteristics as the real-life case.

- **L4 - L8:** These instances are all variations on the real-life case, which are created in the following way: for each pair of a consumption port and an end product in the new instance, we pick the consumption data, along with the corresponding inventory bounds, of a random port-product pair from the real-life case. Note that this can mean that the port does not consume this product in the new instance, if the port we

picked does not store the chosen product in the original instance. In this case, the consumption patterns in the new instances still are realistic in shape and size, but the total demand of a certain port or product may deviate from the total demand in the original case. In this way we create realistic instances, that still have the same characteristics as the original case.

7.3. Exact solution methods

In order to compare the two mathematical formulations of MIP and CP, we set up a computational experiment. In this experiment, we model the maritime inventory routing problem as both a MIP model, as presented in Section 6.2, and a CP model, which is formulated in Section 6.3. We use this model and try to solve it using solution methods that are implemented in commercial solver software. When we use the term exact solution method, we refer to these search algorithms. The commercial solver that we use to solve the MIP model is IBM ILOG CPLEX 12.10.0. To solve the CP model, we use the related solver IBM ILOG CP Optimizer 12.10.0[1]. All experiments are run on a Dell 7490 Windows PC with a 1.90 GHz four-quad core processor and 16 GB RAM.

In this section, this numerical analysis is outlined. First, in Section 7.3.1, the two models are compared based on their size. In Section 7.3.2, we outline the solution approaches that are embedded in the commercial solvers. Lastly, in Section 7.3.3, we discuss the results of the comparison of the exact solution method.

7.3.1. Model size

In this section, we compare the size of the MIP model to the size of the CP model, based on the number of constraints and variables that it contains. Table 7.3 shows the number of variables and constraints for both models for all instances. The size of the models grows fast with increasing instance size. For example, the variables q^s_{itck} in the MIP model are indexed with the number of ports, compartments and products. With doubling the number of ports, compartments and products in the instance, the number of variables of this kind already increase by a factor 8. This shows that the scalability of the model might quickly become an issue when solving for large size instances.

One difference between MIP and CP follows from the decision variables that are related to the routing of the ship. The MIP model uses decision variables of the form x^s_{ijt} for this. Therefore, the number of routing decision variables is quadratic in the number of ports in the instance. Conversely, the CP uses optional interval variables of the form x^s_{ij}, which

[1]A detailed overview of all available CP solvers is available on the following website: http://openjvm.jvmhost.net/CPSolvers/.

is linear in the number of ports in the instance. Therefore, when doubling the number of ports, the number of routing decision variables doubles as well, instead of the quadratic growth in the MIP model.

As explained in Section 4.2.2, the solver IBM ILOG CP Optimizer contains a presolve functionality, which reduces the size of the CP model before starting the search algorithm, by removing redundant constraints and variables. Similarly, the solver for the MIP model, IBM ILOG CPLEX contains a similar functionality, which further reduces the size of the MIP model. In Table 7.4, the size of the models after presolve is reported. What stands out, is that the presolve for the MIP model removes more variables and constraints from the model than the presolve for the CP model. However, even after presolve, the total number of variables in the MIP model is higher than the number of variables in the CP model, up to around a factor three for the large instances.

Table 7.3: In this table, we list the number of variables and constraints for all instances with time horizon $T = 20$. We indicate the number of variables that are in the model formulation, so before presolve is executed.

	MIP			CP	
	Constraints	Binary variables	Total variables	Constraints	Total variables
S1	13748	2056	8888	3117	2096
S2	7812	1506	5510	1977	1676
S3	9274	1692	6336	2193	1636
M1	18450	2510	11870	4343	3277
M2	23778	3076	15154	5559	4296
M3	20271	2700	12996	4790	3585
M4	33538	4272	21336	7792	6236
M5	80662	7350	47286	16680	11030
L1-L8	277038	16114	150738	51729	28253
Case study	277038	16114	150738	51729	28253

Table 7.4: In this table, we list the number of variables and constraints for all instances with time horizon $T = 20$. We indicate the number of variables after presolve is executed by the solver.

	MIP			CP	
	Constraints	Binary variables	Total variables	Constraints	Total variables
S1	2318	850	2203	3117	1914
S2	1526	658	1334	1977	1578
S3	1849	730	1476	2193	1514
M1	3158	1152	3450	4343	3083
M2	4069	1414	3412	5559	4084
M3	3627	1194	3263	4790	3373
M4	5812	2190	4516	7792	6006
M5	10834	3642	8959	16680	10548
L1	36334	9244	31299	51729	26895
L2	37874	9400	32695	51729	26895
L3	36151	9088	31260	51729	26895
L4	45507	8932	40788	51729	26895
L5	47079	8932	42360	51729	26895
L6	45895	8932	41183	51729	26895
L7	45202	8932	40482	51729	26895
L8	45996	8932	41285	51729	26895
Case study	34716	8932	29969	51729	26895

7.3.2. Search algorithms

In this section, we briefly address the exact solution methods in the solvers for the MIP and the CP model. We formulate both models and use the solution algorithms in their respective solvers to solve this problem. This solution process is run until either the instance is solved to optimality, or a time limit of 3600 seconds is reached.

The solution algorithm in IBM ILOG CPLEX is called the *dynamic search* algorithm. As this is a commercial software package, the details of this algorithm are not exactly known. The algorithm is based on the branch and bound algorithm that was outlined in Chapter 3. This algorithm is run with the default settings in CPLEX.

The performance of the automatic search algorithm for the CP model formulation as given in Section 6.3 is analysed. The search algorithm for the constraint programming model is explained in more detail in Section 4.2.2. We guide this search algorithm by the de-

claration of a *search phase*. Experimentation showed that without the declaration of such search phase, there is a high risk that the solution algorithm makes a bad branching decision, from which it is unable to recover soon enough. Our search phase consists of the following variables:

x^s_{ij}	Optional interval variables for the port visit to port i during the j-th voyage of ship s, for $j \in \mathcal{J}^s$
q^s_{ijck}	Discrete variable of the quantity of product $k \in \mathcal{K}^E$ that is loaded into or from compartment $c \in \mathcal{C}^v$ of ship $s \in \mathcal{S}$ at the visit to port $i \in \mathcal{P}$ during the j-th voyage, for $j \in \mathcal{J}^s$
z_{ikt}	Discrete variable for the excess inventory level of base product $k \in \mathcal{K}^B$ at the refinery r at time $t \in \mathcal{T}$, or for the shortage of end product $k \in \mathcal{K}^E$ at consumption port $i \in \mathcal{P}^C$ at time $t \in \mathcal{T}$

Next to adding the variables list above to the search phase, a value selector was included for the penalty variables z_{ikt}, requiring that the smallest possible value was chosen for these variables, in order to minimize the impact on the objective function.

7.3.3. Results

In this section, we discuss the results of the computational study using commercial solvers. The main objective of this study is to compare the results of the solution algorithms in commercial solvers for the MIP and CP model. We compare the results of the solution approach in multiple ways. First, we compare the solutions with the best objective values after the search procedure is finished. Next to that, we analyse the first feasible solutions, as well as the first solutions that reach a certain threshold value in the objective function. Lastly, we compare the time that was needed for both solution approaches to reach a solution of comparable quality. All analyses that we performed are explained in more detail in the following sections.

Best feasible

First, we perform a run time analysis to compare the performance of the commercial solvers for MIP and CP. For all the instances that are created, we run the built-in solution algorithm in order to find the optimal solution. In case this optimal solution is not found, the search is cut off after a time limit of 3600 seconds.

We analyse the runs for two time horizons, namely $T = 20$ days and $T = 60$ days. The reason we chose these time horizons is as follows: we want to compare for both a short and a longer time horizon. Experimentation showed that for a planning horizon of 20 days both solution approaches are still able to find an optimal solution within the time limit. We need this proof of optimality to have a reference of the quality of our results. Next to

Table 7.5: This table contains the results of the run time analysis for small and medium-sized instances. The analysis is done for time horizons $T = 20$ and $= 60$ days. For each model, the best solution is shown, as well as the time it took the solver to reach this solution.The solution process has a time limit of 3600 seconds.

		MIP				CP		
Instance	Horizon	Best solution	Bound	Gap	Time (s)	Best solution	Bound	Time (s)
S1	T=20	**762**	762	0%	4.45	**762**	0	274.07
	T=60	**3146**	1551	50.68%	3596.77	3824	0	2193.79
S2	T=20	**849**	849	0%	0.39	**849**	0	1.0
	T=60	1616631	11820	99.27%	5.17	**1026593**	570775	688.63
S3	T=20	**525**	525	0%	0.70	**525**	0	0.88
	T=60	**1365**	1174	13.93%	3313.72	54463	0	1152.5
M1	T=20	**1146**	873	23.81%	1791.95	1160	0	12.05
	T=60	64170	1212	98.11%	3570.91	**31667**	0	2374.42
M2	T=20	**1528**	1528	0%	1567.45	1852	0	59.84
	T=60	1616269	3788	99.77%	3502.28	**2106664**	546574	1560.73
M3	T=20	**663**	663	0%	327.63	768	0	161.41
	T=60	**273939**	258730	5.55%	3173.08	879315	0	1320.29
M4	T=20	**56758**	1728	96.95%	3534.61	133114	0	3210.51
	T=60	5603470	11873	99.79%	1674.55	**5375996**	401615	2175.48
M5	T=20	21143	1148	94.57%	1321.88	**2693**	0	146.94
	T=60	10619929	957	99.98%	1145.92	**3268506**	0	2206.93

that, the analysis for the longer time horizon of $T = 60$ allows us to test the scalability of both models in terms of an increasing time horizon, as well as to compare the results of a rolling-horizon heuristic, as is explained in Section 7.4.

In Table 7.5 the results of the run time analysis are listed for the small and medium-sized instances. For the three small-sized instances, both the MIP and CP approach found the optimal solution within the time limit for the short planning horizon of $T = 20$. For instances M2 and M3, the MIP approach also found an optimal solution within the time limit. It stands out from this table that for small instances and the short time horizon, MIP is the dominant approach, finding a better solution for 7 out of the 8 instances. For the longer planning horizon of $T = 60$, the model sizes grow and CP finds better quality solutions for most instances. One drawback of the constraint programming approach becomes clear in this table, namely the fact that the CP solver often does not generate good bounds.

Table 7.6: This table contains the results of the run time analysis for large-sized instances. The analysis is done for time horizons T = 20 and = 60 days. For each model, the best solution is shown, as well as the time it took the solver to reach this solution.The solution process has a time limit of 3600 seconds.

		MIP				CP		
Instance	Horizon	Best solution	Bound	Gap	Time to best (s)	Best solution	Bound	Time to best (s)
L1	T=20	**4473**	1140	74.50%	3368.58	17371	0	2950.56
	T=60	29481186	1136	100.00%	263.75	**2876510**	0	2578.27
L2	T=20	33558	1377	95.90%	3537.64	**3959**	0	3179.62
	T=60	29481186	959	100.00%	292.80	**1226275**	0	2398.54
L3	T=20	**4584**	1062	76.82%	3306.23	8946	0	3192.55
	T=60	20507334	1322	99.99%	3510.64	**1958668**	0	3169.36
L4	T=20	**8396**	3601	57.10%	3591.88	75641	0	708.23
	T=60	21171390	288921	98.64%	922.03	**4126265**	0	2987.46
L5	T=20	**111690**	4724	95.77%	3536.72	353604	0	3436.45
	T=60	27849275	169	100.00%	3600.02	**9099865**	0	2953.16
L6	T=20	**34885**	2564	92.65%	2548.41	50789	0	2547.02
	T=60	15237129	517494	96.60%	572.08	**3228286**	0	2647.25
L7	T=20	**1752**	1193	31.88%	3251.81	126210	0	1361.21
	T=60	23448125	363116	98.45%	1142.47	**5068536**	0	3531.31
L8	T=20	**22285**	2125	90.46%	3561.95	107766	0	339.6
	T=60	17735884	407	100.00%	3600.28	**4018831**	0	3324.29
Case study	T=20	19939	2089	89.52%	3543.27	**16983**	0	3215.38
	T=60	26960315	1403	99.99%	3265.58	**1508203**	0	2915.33

When the instance sizes become larger, and therefore the the complexity of the model is increases, the CP-based approach provides superior results. One of the reasons for this could be the fact that in the CP solver, constraint propagation becomes a powerful technique that enables the solver to keep finding better solutions. For many of the large instances, the MIP approach is not able to find a feasible solution within the time limits, for the time horizon of T=60.

Comparable solutions

In the previous analysis, the two solution approaches were compared to each other based on the objective value of the solution that was obtained after the solution period of 3600 seconds. Another method to compare the two approaches is by looking at the time that both approaches to reach a solution with a comparable objective value. For this, we take the minimum objective value that is reached by either the MIP or CP approach, and list the time that it took for both approaches to reach this solution. The results of this comparison

are listed in Tables 7.7 and 7.8.

For a short planning horizon, there is no difference to be distinguished between the two approaches. However, for longer planning horizon of $T = 60$, it is easy to see that for all large instances, the constraint programming approach found a solution that has a similar or better objective value than the MIP solution in only a couple of seconds. The mechanism to construct solutions early in the solution process is one of the benefits of opting for a CP approach.

Table 7.7: This table contains time to a comparable solution for a planning horizon of $T = 20$.

Instance	Horizon	Value MIP	Time MIP (s)	Value CP	Time CP (s)
S1	T=20	762	**4.45**	762	274.07
S2	T=20	849	**0.39**	849	1.00
S3	T=20	525	**0.7**	525	0.88
M1	T=20	1156	291.92	1160	**12.05**
M2	T=20	1530	1433.14	1852	**59.84**
M3	T=20	718	**108.48**	768	161.41
M4	T=20	126859	**10.78**	133114	1198.39
M5	T=20	21143	1321.88	20889	**120.71**
L1	T=20	15024	**1518.50**	17371	2950.56
L2	T=20	33558	3537.64	33382	**110.96**
L3	T=20	8577	3302.58	8946	**1614.99**
L4	T=20	49108	907.97	75641	**708.23**
L5	T=20	326989	**448.56**	353604	2397.5
L6	T=20	49796	**977.42**	50789	2547.02
L7	T=20	125259	**319.63**	126210	1361.21
L8	T=20	91687	1788.86	107766	**339.6**
Case study	T=20	19939	3543.27	19929	**1320.36**

Table 7.8: This table contains time to a comparable solution for a planning horizon of $T = 60$.

Instance	Horizon	Value MIP	Time MIP (s)	Value CP	Time CP (s)
S1	T=60	3621	3458.55	3824	**2193.79**
S2	T=60	1616631	5.17	1441227	**3.97**
S3	T=60	45993	**2.94**	54463	1152.5
M1	T=60	64170	3570.91	63941	**212.60**
M2	T=60	2097763	**1171.97**	2106664	1560.73
M3	T=60	809914	**232.45**	879315	1320.29
M4	T=60	5603470	1674.55	5591328	**1155.61**
M5	T=60	10619929	1145.92	10603004	**36.25**
L1	T=60	29481186	263.75	29481186	**20.94**
L2	T=60	29481186	292.8	29481186	**9.73**
L3	T=60	20507334	3510.64	9887118	**95.26**
L4	T=60	21171390	922.03	21171390	**10.53**
L5	T=60	27849275	3600.02	27849275	**12.25**
L6	T=60	15237129	572.08	15237129	**22.14**
L7	T=60	23448125	1142.47	23448125	**10.37**
L8	T=60	17735884	3600.28	17735884	**10.62**
Case study	T=60	26960315	3265.58	23601350	**20.62**

First feasible

Apart from comparing the best solutions that we found in the run time analysis, there is some information in the the time until a first feasible solution is found. In Table 7.9, we analyse the results from the analysis in the previous section by the time to the first feasible solution. For small instances, the time until a first feasible solution is found, is comparable. As the instance sizes and time horizon increase, it is easy to observe that for MIP, it takes more time to find an initial solution. The reason for this, is that for these large instances, the MIP model becomes larger, making it harder to solve the root node. The CP solver suffers less from this scalability issue, as it is able to construct a feasible solution relatively quickly.

Table 7.9: This table contains the time to the first feasible solutions for all instances.

Instance	Horizon	Time MIP (s)	Time CP (s)	Instance	Horizon	Time MIP (s)	Time CP (s)
S1	T=20	**0.17**	0.64	L1	T=20	20.52	**2.87**
	T=60	**0.78**	1.14		T=60	263.70	**15.83**
S2	T=20	**0.13**	0.18	L2	T=20	24.69	**2.92**
	T=60	0.83	**0.32**		T=60	292.75	**7.15**
S3	T=20	**0.13**	0.20	L3	T=20	19.51	**2.94**
	T=60	0.66	**0.29**		T=60	259.91	**17.24**
M1	T=20	0.28	**0.25**	L4	T=20	49.99	**3.11**
	T=60	9.56	**0.48**		T=60	921.98	**7.69**
M2	T=20	**0.58**	1.14	L5	T=20	44.80	**3.07**
	T=60	2.41	**0.58**		T=60	22059.91	**8.58**
M3	T=20	0.41	**0.29**	L6	T=20	39.58	**3.25**
	T=60	1.88	**0.51**		T=60	572.08	**16.19**
M4	T=20	0.83	**0.39**	L7	T=20	34.59	**3.08**
	T=60	4.64	**1.76**		T=60	1142.42	**7.59**
M5	T=20	2.23	**0.59**	L8	T=20	57.50	**3.22**
	T=60	15.34	**2.97**		T=60	3600.25	**7.68**
				Case study	T=20	18.39	**2.83**
					T=60	254.89	**11.23**

7.4. Rolling-horizon heuristic

In the previous section, we analysed the scalability of the maritime inventory routing problem and found that for large instances, the solution methods in the commercial solvers were not only not able to find an optimal solution within the search limits of our experiment, but already finding good quality solutions proved to be a hard task. However, in a practical setting, the aim is to construct solutions for a time period of several months. Therefore, it is desirable to find a solution method that can come up with such solutions. For this reason, we develop a construction heuristic method in order to come up with good quality solutions within a similar time frame. In this section, we explain the principles behind our heuristic and set up an experiment to evaluate the performance of the heuristic.

7.4.1. Setup

It followed from the computational experiment in the previous section, that when increasing the time scale, both the MIP and CP approaches had trouble finding good quality solutions. Papageorgiou, Cheon et al. (2018) discuss the use of matheuristics for the maritime inventory routing problem. The authors list several types of construction heuristics, most notably rolling-horizon heuristics and local search heuristics. The local search heuristics in this paper are more suitable for problems with a large fleet size, as they rely on solving the problem for a subset of one or two ships out of the entire fleet. As the fleet only exists of two ships in the instances in this thesis, we opt for the use of a rolling-horizon heuristic. The objective of this heuristic is to find high quality solutions for a mathematical program, by iteratively fixing a subset of variables in the model. In this section, we provide a framework in which both MIP and CP can be used in a heuristic approach.

In a rolling-horizon heuristic, the problem is solved for the desired planning horizon by iteratively solving smaller problems with a shorter planning horizon and extending the planning horizon with every iteration. After every iteration, part of the solution is fixed for the next iteration. In this way, the sizes of the model does not grow as hard as it would when considering the entire model.

Algorithm 1 gives the outline of the rolling-horizon heuristic to solve an instance $\mathcal{I}_T$ of the maritime inventory routing problem that has planning horizon T. The iteration horizon τ determines the size of the models that are solved in each iteration. In order to limit the running time of the solution processes, the solve timit limit $t_{\max}$ is predetermined. The horizon of the first iteration is initialised to be τ.

One iteration of the heuristic is described in lines 11-24. For iteration k, an instance $\mathcal{I}_{T_k}$ with a time horizon T_k is created, for which a mathematical model $\mathcal{M}_k$ is built. This model can be a MIP or CP model, as formulated in Chapter 6. For $k = 0$, this model is solved using

Algorithm 1: Rolling-horizon heuristic for the maritime inventory routing problem

1 **Input**
2 Problem instance $\mathcal{I}_T$ for planning horizon T, and number of ships S
3 Iteration horizon τ
4 Solve time limit per iteration $t_{\max}$
5
6 **Initialize**
7 $k = 0$
8 Planning horizon for first iteration $T_0 = \tau$
9
10 **while** $T_k < T$ **do**
11 Create instance $\mathcal{I}_{T_k}$
12 Build mathematical model $\mathcal{M}_k$ for instance $\mathcal{I}_{T_k}$
13
14 **if** $k > 0$ **then**
15 **for** All ships $s = 1, \ldots, S$ **do**
16 Add constraints to fix the first k voyages from Σ_{k-1} in $\mathcal{M}_k$
17 Add constraints to fix corresponding (un)loading quantities in $\mathcal{M}_k$
18
19 Solve $\mathcal{I}_{T_k}$ within time limit $t_{\max}$, resulting in solution Σ_k
20 **for** All ships $s = 1, \ldots, S$ **do**
21 Extract list of voyages $V_{k,s} = \{v_{k,s,1}, \ldots, v_{k,s,m}\}$ from solution Σ_k
22
23 $T_{k+1} = \min \left\{ \min_{\text{ships } s} \left\{ \texttt{endOf}(v_{k,s,k+1}) \right\} + \tau, T \right\}$
24 $k \leftarrow k + 1$
25
26 Do one last iteration as described in lines 11-24 for $T_k = T$

the same techniques as used in the analysis earlier in this chapter. This solution process has a runtime limit of $t_{\max}$. The solution Σ_0 of this process is stored. From the routing variables in this solution, a list of voyages is deduced. Remember from the earlier definition, that a voyage encompasses exactly one round trip from and to the refinery. This information is used to reduce the model size in the next iteration. The time that ships s return to the refinery after their first voyage is denoted as

$$\min_{\text{ships } s} \left\{ \texttt{endOf}(v_{k,s,k+1}) \right\}$$

with a maximum of T. The planning horizon T_1 for the next iteration is set as this minimum plus the iteration horizon τ, effectively resulting in a new planning period of τ.

For $k > 0$, a similar procedure is executed. After building model $\mathcal{M}_k$ for instance $\mathcal{I}_{T_k}$

with time horizon T_k, the first k voyages for both ships in Σ_{k-1} are fixed in $\mathcal{M}_k$, including the quantities that are loaded or unloaded during these voyages. In the MIP model, this corresponds to fixing the routing variables x^s_{ijt} corresponding to this voyage. In the CP model, the corresponding port stops are fixed. The reason that the (un)loading quantities are fixed, is to prevent the solution process from reoptimizing these quantities repetitively during each following iteration. Once the time horizon T_k has been set to the total planning horizon T, one last iteration is performed to solve the problem for the desired time horizon.

7.4.2. Results

In this section, we analyze the performance of the rolling-horizon heuristic that was given by Algorithm 1. The algorithm builds and solves mathematical models for a given problem instance with time horizon T. We use both MIP and CP as part of the modelling of this heuristic and compare the results. In our analysis, we use a planning horizon of $T = 60$ and an iteration horizon $\tau = 20$, and compare the results to the results of the analysis of the exact solution methods in Section 7.3.3. An iteration solve limit of 600 seconds is used.

The results for the heuristic with the MIP formalism at its core are listed in Table 7.10. In this table, the objective values of the first iteration of the rolling-horizon heuristic, i.e. with a planning horizon of $T = 20$, and the last iteration, with a planning horizon of $T = 60$, are compared to the results in Table 7.5 and Table 7.6. Similarly, Table 7.11 contains the results for the heuristic procedure with CP at its core. From Table 7.10, it follows that the rolling-horizon heuristic is able to find better solutions that the exact solution method, especially for large-sized instances. With the number of iterations that is between 4 and 7 for most instances, the total run time for the heuristic is comparable to the run time that was given to the exact solution method. It follows that the approach to reduce the model size of the instances that we solve in each iteration yields better results than solving the complete model directly for the MIP-based approach.

In Table 7.11, the results for the CP-based rolling-horizon heuristic are listed. Especially for the smaller instances, the heuristic is able to find better solutions than the exact solution method. However, for larger instances, the results are more mixed than in the case of the MIP-based approach. The reason for this is twofold. On the one hand, the CP-based exact solution method already provided better solutions, making it a tougher benchmark to match. Additionally, the decision to fix the loading quantities was made in order to reduce computation time in the CP-based heuristic approach, as the CP solver spends a lot of time to optimise the loading and unloading quantities.

Table 7.10: This table contains the results of the runs for the rolling-horizon heuristic based on mixed integer programming. The table shows the results for the first iteration, after $T = 20$, and at the end of the planning horizon, at $T = 60$ days. For each iteration, the solution process has a time limit of 600 seconds. The solutions that are listed in the columns with headers 'original', are the solutions that were found using the direct solution approach as explained in Section 7.3.

	T=20		T=60			
Instance	Heuristic	Original	Heuristic	Original	Bound	# iterations
S1	762	762	**2900**	3146	1551	4
S2	849	849	85792	**53464**	12621	7
S3	525	525	**968**	1365	1174	2
M1	1152	1146	**4142**	64170	1212	5
M2	1852	1528	**900292**	1616269	3787	7
M3	663	663	279606	**273939**	258489	7
M4	72899	56758	**1829455**	5603470	11844	7
M5	17387	21143	**1203426**	10619929	1782	7
L1	89575	4473	**535871**	29481186	212	4
L2	76927	33558	**12757**	29481186	71	6
L3	77845	4584	**257495**	20507334	1322	5
L4	137583	8396	**755658**	21171390	1053	4
L5	247917	111690	3584320	**27849275**	169	5
L6	85544	34885	**795315**	15237129	0	5
L7	81737	1752	**950970**	23448125	78	5
L8	103135	22285	**975821**	17735884	407	4
Case study	155891	19939	**473026**	26960315	1403	5

Table 7.11: This table contains the results of the runs for the rolling-horizon heuristic based on constraint programming. The table shows the results for the first iteration, after $T = 20$, and at the end of the planning horizon, at $T = 60$ days. For each iteration, the solution process has a time limit of 600 seconds. The solutions that are listed in the columns with headers 'original', are the solutions that were found using the direct solution approach as explained in Section 7.3.

	T=20		T=60		
Instance	Heuristic	Original	Heuristic	Original	# iterations
S1	1034	762	3917	**3824**	4
S2	849	849	**41603**	1026593	6
S3	525	525	**19045**	54463	6
M1	1156	1160	**31210**	31667	5
M2	17465	1852	**1548283**	2106664	6
M3	932	768	5482252	**879315**	6
M4	129911	133114	**2196188**	5375996	6
M5	90312	2693	5321157	**3268506**	5
L1	6509514	17371	**2344633**	2876510	4
L2	104343	3959	2436266	**1226275**	6
L3	116159	8946	2975490	**1958668**	5
L4	223913	75641	4666873	**4126265**	4
L5	431032	353604	**8473602**	9099865	4
L6	153236	50789	3265499	**3228286**	4
L7	213081	126210	6798315	**5068536**	4
L8	107766	107766	**2677910**	4018831	5
Case study	29804	16983	**799122**	1508203	5

Table 7.12: This table compares the results of the runs for the rolling-horizon heuristic based on mixed integer programming and constraint programming. The table shows the results for the first iteration, after $T = 20$, and at the end of the planning horizon, at $T = 60$ days. For each iteration, the solution process has a time limit of 600 seconds.

	T=20			T=60	
Instance	MIP	CP	Instance	MIP	CP
S1	**2900**	3917	L1	**535871**	2344633
S2	85792	**41603**	L2	**12757**	2436266
S3	**968**	19045	L3	**257495**	2975490
M1	**4142**	31210	L4	**755658**	4666873
M2	**900292**	1548283	L5	**3584320**	8473602
M3	**279606**	5482252	L6	**795315**	3265499
M4	**1829455**	2196188	L7	**950970**	6798315
M5	**1203426**	5321157	L8	**975821**	2677910
			Case study	**473026**	799122

After comparing the MIP-based and CP-based heuristic to the exact solution methods in Tables 7.10 and 7.11 respectively, the methods are compared to each other in Table 7.12. For all but one instance, the MIP-based approach found better results. The main reason for this difference is that the CP-based approach suffers more from the limited solution time. Errors that are made in early iterations cannot be recovered in later iterations for longer planning horizons.

8

Discussion

In this thesis, two different solution approaches for the maritime inventory routing problem were compared. The aim of the thesis was to provide answers to three main research questions, which are:

1. How can the maritime inventory routing problem be modelled using mixed integer programming (MIP) and constraint programming (CP)?
2. How does the performance of the solution algorithms in commercial solver software for the MIP and CP formulations of the maritime inventory routing problem compare?
3. How does the performance of the rolling-horizon heuristics for the MIP and CP formulations of the maritime inventory routing problem compare?

In this chapter, we discuss the findings of this research related to these questions and discuss possible improvements that can be made in future research.

In Chapter 6, both a MIP and CP model were formulated for the maritime inventory routing problem. The aim was to provide two models that are equivalent to each other, meaning that all feasible solutions for one of the two models has an equivalent solution in the other model, without resulting in infeasibilities. To the best of our knowledge, the two models provided in Chapter 6 are equivalent to each other, with one exception, namely the use of voyages in the constraint programming model. The choice to use explicitly formulate voyages in this model, was driven by the aim to reduce the possibilities to schedule port stops, helping the solution process in finding good solutions. This voyage formulation could also be implemented in the mixed integer model, albeit less straightforwardly than in the constraint programming model. Therefore, it could theoretically be possible

that a solution to the MIP model visits a consumption port multiple times before ending its voyage at the refinery. In our computational study however, we have not encountered solutions that leaded to this kind of infeasiblity, as it is in practice suboptimal to visit a port twice during one voyage.

The aim of this thesis was to compare the performance of the solution approaches for the two model formulation as directly as possible. The MIP model was therefore formulated without the use of valid inequalities. However, the use of these inequalities can improve the performance of the MIP solution approach. For example, Foss et al. (2016) developed valid inequalities for a maritime inventory routing product with undedicated compartments.

Similarly, possible improvements for the constraint programming can also be made. For example, the possibility to define custom constraints with accompanying filtering procedures offers a chance to explicitly use information about the instance of a problem in the solution process. From production and consumption information at all ports, it might be possible to induce time windows for port visits, reducing the domains for these variables. This and other improvements to the model might be worth exploring in future research. The Python API for the constraint programming solver that was used when conducting this thesis, does not support this kind of manual extensions to the model, so it is recommended to choose a solver and programming language that does support these kind of additions.

Another improvement that could strengthen the constraint programming model formulation, is replacing the soft inventory bounds constraints by hard constraints. The constraint programming solver contains a filtering algorithm for the `alwaysIn` constraint, which moves interval variables in order to find solutions that satisfy this hard constraint. However, when this constraint is modelled as a soft constraint, by introducing the slack variables to the model, these filtering algorithms become ineffective, as it is always possible to find a feasible solution, by choosing the right value for the slack penalty. Nevertheless, due to the nature of the instances that are included in this thesis, the use of soft constraints was inevitable, as posing the bounds as hard constraints would lead to infeasible models. It can be interesting to formulate the model using hard constraints for suitable instances and analyse the performance of the models.

In the computational study in Chapter 7, the aim was to make a direct comparison between the solution approaches for the MIP and CP modelling paradigm, using commercially available solvers. The most important takeaways are that for smaller instances, the MIP model performs better, but with increasing instance size, CP found better solutions.

There are some possible improvements to the approach that was taken in this thesis. First of all, we only provide a direct comparison between two possible solution approaches. In most of the cases, an optimal solution was not found, so nothing can be said about the

absolute strength of either solution approach. The use of better lower bounds could enhance the discovery of optimal solutions or at least provide a better threshold to the objective value. Next to that, the instances can be created with more care, so it is easier to weigh the value of a solution that is found. Instances for which a solution exists that does not have penalties for inventory bound violations have much lower objective values, making it easier to interpret the quality of solutions based on their objective value.

With the development of the rolling-horizon heuristic, we provide a tool that is more effective in finding good quality solutions. Nevertheless, the same argument as in the previous paragraph extends to the analysis of the heuristic. Even though a relative comparison between the heuristics using a MIP and CP approach can be made, we lack knowledge about the desired result, making it hard to evaluate the true potential of this heuristic. One thing that can be concluded, is that the heuristic is more effective using the MIP approach, as the heuristic iteration solve limit of 600 seconds is too short for the constraint programming model to find a good solution to large-sized instances.

In the heuristic, the choice was made to fix the loading variables from the solution of one iteration to model in the next iteration. However, this decision may lead to the fact that, by fixing these quantities, the heuristic becomes less flexible when the planning horizon is shifted. New information that comes available in a next iteration, might change the optimal allocation of products to tanks and ports. This flexibility is reduced by fixing the quantities up front. On the other hand, not fixing these variables might lead to worse solutions for the constraint programming heuristic. This is a trade-off that is needed to be made.

Both in the analysis of the exact solution method and the rolling-horizon heuristic, the MIP and CP approaches have been compared vis-a-vis. However, it might be beneficial to not treat both paradigms as stand-alone solvers, but integrate the two approaches in order to profit from the strengths of both methods. A first step could be to use the propagation techniques from constraint programming to find feasible schedules for the problem, and subsequently solve the problem of loading and unloading quantities using the LP mechanism in the MIP solver. This and more sophisticated integrated methods could be an interesting topic in future research.

A

Model formulations

A.1. Mixed integer programming model

$$\text{minimize} \sum_{s\in\mathcal{S},\,(it,j\tau)\in\mathcal{A}^s} C^s_{ij} x^s_{(it,j\tau)} + \sum_{i\in\mathcal{P},\,k\in\mathcal{K},\,t\in\mathcal{T}} \alpha_{ikt} z_{ikt}. \tag{A.1}$$

s.t.

$$\begin{aligned}
x^s_{o^s r1} + x^s_{o^s d^s} &= 1, && \forall s\in\mathcal{S}, && \text{(A.2)}\\
\sum_{t\in\mathcal{T}} x^s_{rd^s t} + x^s_{o^s,d^s} &= 1, && \forall s\in\mathcal{S}, && \text{(A.3)}\\
\sum_{a\in\mathcal{A}^+_n} x^s_a &= \sum_{a\in\mathcal{A}^-_n} x^s_a, && \forall s\in\mathcal{S}, n\in\mathcal{N}^s\setminus(o^s\cup d^s), && \text{(A.4)}\\
x^s_{rrt} &= 1, && \forall s\in\mathcal{S}, t\in[s_u,e_u), u\in U^s, && \text{(A.5)}\\
x^s_{ijt} &\in \{0,1\}, && \forall (it,j\tau)\in\mathcal{A}^s, s\in\mathcal{S}, && \text{(A.6)}\\
l^s_{tck} &= l^s_{(t-1)ck} + q^s_{rtck} - \sum_{i\in\mathcal{P}^C} q^s_{itck}, && \forall s\in\mathcal{S}, t\in\mathcal{T}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.7)}\\
o^s_{it} &= \sum_{j\in\mathcal{P}} x^s_{ij(t-O^s_i)}, && \forall s\in\mathcal{S}, i\in\mathcal{P}, t\in\mathcal{T}, && \text{(A.8)}\\
q^s_{itck} &\geq \underline{Q}_{ik}\, o^s_{it}, && \forall s\in\mathcal{S}, i\in\mathcal{P}, t\in\mathcal{T}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.9)}\\
q^s_{itck} &\leq M\, o^s_{it}, && \forall s\in\mathcal{S}, i\in\mathcal{P}, t\in\mathcal{T}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.10)}\\
l^s_{tck} &\leq \overline{L}^s_{ck}(1-\sum_{i\in\mathcal{P}} x^s_{ir(t-O^s_i)}), && \forall s\in\mathcal{S}, t\in\mathcal{T}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.11)}\\
l^s_{0ck} &= 0, && \forall s\in\mathcal{S}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.12)}\\
0 &\leq l^s_{tck} \leq \overline{L}^s_{ck}, && \forall s\in\mathcal{S}, t\in\mathcal{T}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.13)}\\
q^s_{itck} &\geq 0, && \forall s\in\mathcal{S}, i\in\mathcal{P}, t\in\mathcal{T}, c\in\mathcal{C}^s, k\in\mathcal{K}^E, && \text{(A.14)}\\
o^s_{it} &\in \{0,1\}, && \forall s\in\mathcal{S}, i\in\mathcal{P}, t\in\mathcal{T}, && \text{(A.15)}
\end{aligned}$$

$$l^s_{tck} \leq \overline{L}^s_{ck} y^s_{tck}, \qquad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \quad \text{(A.16)}$$

$$\sum_{k \in \mathcal{K}^E} y^s_{tck} \leq 1 \qquad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \quad \text{(A.17)}$$

$$y^s_{tck} \in \{0,1\}, \qquad \forall s \in \mathcal{S}, t \in \mathcal{T}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \quad \text{(A.18)}$$

$$z_{rkt} = \max\{s_{rkt} - \overline{I}_{rk}, 0\}, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}^B, \quad \text{(A.19)}$$

$$z_{ikt} = \max\{\underline{I}_{ik} - s_{ikt}, 0\}, \qquad \forall i \in \mathcal{N}^C, t \in \mathcal{T}, k \in \mathcal{K}^B, \quad \text{(A.20)}$$

$$s_{rkt} = s_{rk(t-1)} - \sum_{s \in \mathcal{S}, c \in \mathcal{C}^s, \kappa \in \mathcal{K}^E} F_{k\kappa} q^s_{rtc\kappa} + R^P_{rkt}, \qquad \forall k \in \mathcal{K}^B, t \in \mathcal{T}, \quad \text{(A.21)}$$

$$s_{ikt} = s_{ik(t-1)} + \sum_{s \in \mathcal{S}, c \in \mathcal{C}^s} q^s_{itck} - R^C_{ikt}, \qquad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}, \quad \text{(A.22)}$$

$$z_{rkt} \geq s_{rkt} - \overline{I}_{rk}, \qquad \forall k \in \mathcal{K}^B, t \in \mathcal{T}, \quad \text{(A.23)}$$

$$z_{ikt} \geq \underline{I}_{ik} - s_{ikt}, \qquad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}, \quad \text{(A.24)}$$

$$s_{ik0} = I_{ik0}, \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}, \quad \text{(A.25)}$$

$$s_{rkt} \geq 0, \qquad \forall k \in \mathcal{K}^B, t \in \mathcal{T}, \quad \text{(A.26)}$$

$$s_{ikt} \leq \overline{I}_{ik}, \qquad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, t \in \mathcal{T}, \quad \text{(A.27)}$$

$$z_{ikt} \geq 0, \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T}. \quad \text{(A.28)}$$

A.2. Constraint programming model

$$\text{minimize} \sum_{s\in\mathcal{S}} TC^s + \sum_{i\in\mathcal{P}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} \alpha_{ikt} z_{ikt}, \tag{A.29}$$

$$\text{where} \quad TC^s = \sum_i C^s_{\theta^s(\pi^s(i-1)),\theta(\pi^s(i))}, \quad \forall s \in \mathcal{S}. \tag{A.30}$$

s.t.

$$\texttt{first}(\pi^s, x^s_0), \qquad \forall s \in \mathcal{S}, \tag{A.31}$$

$$\texttt{last}(\pi^s, x^s_T), \qquad \forall s \in \mathcal{S}, \tag{A.32}$$

$$\texttt{noOverlap}(\pi^s, T^s), \qquad \forall s \in \mathcal{S}, \tag{A.33}$$

$$x^s_0 = [0,1), \qquad \forall s \in \mathcal{S}, \tag{A.34}$$

$$x^s_T \subseteq [1,T), \qquad \forall s \in \mathcal{S}, \tag{A.35}$$

$$x^s_u = [s_u, e_u), \qquad \forall s \in \mathcal{S}, u \in \mathcal{U}^s, \tag{A.36}$$

$$\tilde{q}^s_{ijck} = stepAtEnd\left(x^s_{ij}, \underline{Q}_{ik}, \overline{L}^s_{ck}\right), \qquad \forall s \in \mathcal{S}, i \in \mathcal{P}, j \in \mathcal{J}^S, c \in \mathcal{C}^S, k \in \mathcal{K}^E, \tag{A.37}$$

$$\tilde{l}^s_{ck} = \sum_{j\in\mathcal{J}^s} \tilde{q}^s_{rjck} - \sum_{i\in\mathcal{P}^C}\sum_{j\in\mathcal{J}^s} \tilde{q}^s_{ijck}, \qquad \forall s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, [5pt] \tag{A.38}$$

$$\texttt{alwaysIn}\left(\tilde{l}^s_{ck}, 0, T, 0, \overline{L}^s_{ck}\right), \qquad \forall s \in \mathcal{S}, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{A.39}$$

$$\texttt{alternative}\left(x^s_j, \left\{y^s_{jck}\right\}_{k\in\mathcal{K}^E}\right), \qquad \forall s \in \mathcal{S}, j \in \mathcal{J}^s, c \in \mathcal{C}^s, \tag{A.40}$$

$$\left(q^s_{rjck} > 0\right) \Rightarrow presenceOf\left(y^s_{jck}\right), \qquad \forall s \in \mathcal{S}, j \in \mathcal{J}^s, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{A.41}$$

$$y^s_{jck} \quad \subseteq \{\perp\} \cup [0,T), \qquad \forall s \in \mathcal{S}, j \in \mathcal{J}^s, c \in \mathcal{C}^s, k \in \mathcal{K}^E, \tag{A.42}$$

$$\tilde{R}^P_{rk} = \sum_{t\in\mathcal{T}} stepAt\left(t, R^P_{rk}\right), \qquad \forall k \in \mathcal{K}^B, \tag{A.43}$$

$$\tilde{R}^C_{ik} = \sum_{t\in\mathcal{T}} stepAt\left(t, R^C_{ik}\right), \qquad \forall i \in \mathcal{P}^C, k \in \mathcal{K}^E, \tag{A.44}$$

$$\tilde{q}^s_{rjck} = \sum_{\kappa\in\mathcal{K}^E} F_{k\kappa} \tilde{q}^s_{rjc\kappa}, \qquad \forall k \in \mathcal{K}^B, \tag{A.45}$$

$$\tilde{q}_{rk} = \sum_{s\in\mathcal{S}}\sum_{j\in\mathcal{J}^S}\sum_{c\in\mathcal{C}^S}\sum_{\kappa\in\mathcal{K}^E} F_{k\kappa} \tilde{q}^s_{rjc\kappa}, \qquad \forall k \in \mathcal{K}^B, \tag{A.46}$$

$$\tilde{q}_{ik} = \sum_{s\in\mathcal{S}}\sum_{j\in\mathcal{J}^S}\sum_{c\in\mathcal{C}^S} i\tilde{q}^s_{rjck}, \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}^E, \tag{A.47}$$

$$\tilde{z}_{ikt} = pulse(\zeta_{ikt}, z_{ikt}), \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}, \tag{A.48}$$

$$\tilde{z}_{ik} = \sum_{t \in \mathcal{T}} \tilde{z}_{ikt} \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}, \tag{A.49}$$

$$\zeta_{ikt} \subseteq \{\perp\} \in [t, t+1), \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}^E, t \in \mathcal{T}, \tag{A.50}$$

$$z_{ikt} \in \mathbb{Z}_{\geq 0}, \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}^E, t \in \mathcal{T}, \tag{A.51}$$

$$\tilde{s}_{rk} = \tilde{S}^0_{rk} + \tilde{R}^P_{rk} - \tilde{q}_{rk} - \tilde{z}_{rk}, \qquad \forall k \in \mathcal{K}^B, \tag{A.52}$$

$$\tilde{s}_{ik} = \tilde{S}^0_{ik} - \tilde{R}^P_{ik} + \tilde{q}_{ik} + \tilde{z}_{ik}, \qquad \forall k \in \mathcal{K}^B \tag{A.53}$$

$$\texttt{alwaysIn}(\tilde{s}_{ik}, 0, T, \underline{I}_{ik}, \overline{I}_{ik}), \qquad \forall i \in \mathcal{P}, k \in \mathcal{K}. \tag{A.54}$$

References

[1] A. Agra, H. Andersson, M. Christiansen and L. A. Wolsey. 'A maritime inventory routing problem: Discrete time formulations and valid inequalities.' In: 62 (2013), pp. 297–314.

[2] A. Agra, M. Christiansen and A. Delgado. 'Discrete time and continuous time formulations for a short sea inventory routing problem'. In: Optimization and Engineering 18 (Apr. 2016).

[3] A. Agra, M. Christiansen and A. Delgado. 'Mixed Integer Formulations for a Short Sea Fuel Oil Distribution Problem'. In: Transportastion Science 47 (Feb. 2013), pp. 108–124.

[4] H. Andersson. 'A Maritime Pulp Distribution Problem'. In: INFOR: Information Systems and Operational Research 49.2 (2011), pp. 125–138. eprint: https://doi.org/10.3138/infor.49.2.125.

[5] H. Andersson, A. Hoff, M. Christiansen, G. Hasle and A. Løkketangen. 'Industrial aspects and literature survey: Combined inventory management and routing'. In: Computers & Operations Research 37.9 (2010), pp. 1515–1536.

[6] P. Baptiste, P. Laborie, C. L. Pape and W. Nuijten. 'Chapter 22 - Constraint-Based Scheduling and Planning'. In: Handbook of Constraint Programming. Ed. by F. Rossi, P. van Beek and T. Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006, pp. 761–799.

[7] W. J. Bell, L. M. Dalberto, M. L. Fisher, A. J. Greenfield, R. Jaikumar, P. Kedia, R. G. Mack and P. J. Prutzman. 'Improving the Distribution of Industrial Gases with an On-Line Computerized Routing and Scheduling Optimizer'. In: Interfaces 13.6 (1983), pp. 4–23.

[8] C. Bessiere. 'Chapter 3 - Constraint Propagation'. In: Handbook of Constraint Programming. Ed. by F. Rossi, P. van Beek and T. Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006, pp. 29–83.

[9] A. Bockmayr and J. Hooker. 'Constraint Programming'. In: Discrete Optimization. Ed. by K. Aardal, G. Nemhauser and R. Weismantel. Vol. 12. Handbooks in Operations Research and Management Science. Elsevier, 2005, pp. 559–600.

[10] R. I. Brafman. 'A Simplifier for Propositional Formulas with Many Binary Clauses'. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1. IJCAI'01. Seattle, WA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 515–520.

[11] A. M. Campbell and M. W. Savelsbergh. 'A decomposition approach for the inventory-routing problem'. In: Transportation science 38.4 (2004), pp. 488–502.

[12] A. Cesta and A. Oddi. 'Gaining efficiency and flexibility in the simple temporal problem'. In: Proceedings Third International Workshop on Temporal Representation and Reasoning (TIME '96). 1996, pp. 45–50.

[13] B. V. Cherkassky, A. V. Goldberg and T. Radzik. 'Shortest paths algorithms: Theory and experimental evaluation'. In: Mathematical programming 73.2 (1996), pp. 129–174.

[14] M. Christiansen. 'Decomposition of a Combined Inventory and Time Constrained Ship Routing Problem'. In: Transportation Science 33.1 (1999), pp. 3–16.

[15] M. Christiansen and K. Fagerholt. 'Maritime inventory routing problems'. In: Encyclopedia of Optimization. Ed. by C. A. Floudas and P. M. Pardalos. Boston, MA: Springer US, 2009, pp. 1947–1955.

[16] M. Christiansen, K. Fagerholt, T. Flatberg, Ø. Haugen, O. Kloster and E. Lund. 'Maritime Inventory Routing With Multiple Products: A Case Study From the Cement Industry'. In: European Journal of Operational Research 208 (Jan. 2011), pp. 86–94.

[17] M. Christiansen, K. Fagerholt, B. Nygreen and D. Ronen. 'Ship routing and scheduling in the new millennium'. In: European Journal of Operational Research 228.3 (2013), pp. 467–483.

[18] M. Christiansen and B. Nygreen. 'A method for solving ship routing problems with inventory constraints'. In: Annals of Operations Research 81.0 (June 1998), pp. 357–378.

[19] L. Coelho, J.-F. Cordeau and G. Laporte. 'Thirty Years of Inventory Routing'. In: Transportation Science 48 (Jan. 2015), pp. 1–19.

[20] M. Conforti, G. Cornuéjols, G. Zambelli et al. Integer programming. Vol. 271. Springer, 2014.

[21] A. L. Custódio and R. C. Oliveira. 'Redesigning distribution operations: a case study on integrating inventory management and vehicle routes design'. In: International Journal of Logistics Research and Applications 9.2 (2006), pp. 169–187. eprint: https://doi.org/10.1080/13675560600649982.

[22] G. Dantzig, R. Fulkerson and S. Johnson. 'Solution of a Large-Scale Traveling-Salesman Problem'. In: Journal of the Operations Research Society of America 2.4 (1954), pp. 393–410.

[23] S. Dauzère-Pérès, A. Nordli, A. Olstad, K. Haugen, U. Koester, P. Myrstad, G. Teistklub and A. Reistad. 'Omya Hustadmarmor Optimizes Its Supply Chain for Delivering Calcium Carbonate Slurry to European Paper Manufacturers'. In: Interfaces 37 (Feb. 2007), pp. 39–51.

[24] R. Dechter, I. Meiri and J. Pearl. 'Temporal constraint networks'. In: Artificial Intelligence 49.1 (1991), pp. 61–95.

[25] S. Demassey. Global Constraint Catalog. Accessed: 2020-08-09.

[26] M. Dror and M. Ball. 'Inventory/routing: Reduction from an annual to a short-period problem'. In: Naval Research Logistics (NRL) 34.6 (Dec. 1987), pp. 891–905.

[27] E. Foss, T. N. Myklebust, H. Andersson and M. Christiansen. 'A Multi-product Maritime Inventory Routing Problem with Undedicated Compartments'. In: Computational Logistics. Ed. by A. Paias, M. Ruthmair and S. Voß. Cham: Springer International Publishing, 2016, pp. 3–17.

[28] S. Gay, R. Hartert and P. Schaus. 'Simple and Scalable Time-Table Filtering for the Cumulative Constraint'. In: vol. 9255. Aug. 2015, pp. 149–157.

[29] R. Gedik, C. Rainwater, H. Nachtmann and E. A. Pohl. 'Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals'. In: European Journal of Operational Research 251.2 (2016), pp. 640–650.

[30] K. Giles and W.-J. van Hoeve. 'Solving a Supply-Delivery Scheduling Problem with Constraint Programming'. In: Principles and Practice of Constraint Programming. Ed. by M. Rueher. Cham: Springer International Publishing, 2016, pp. 602–617.

[31] V. Goel, K. Furman, J.-H. Song and A. El-Bakry. 'Large neighborhood search for LNG inventory routing'. In: Journal of Heuristics 18 (Dec. 2012).

[32] V. Goel, M. Slusky, W.-J. van Hoeve, K. Furman and Y. Shao. 'Constraint programming for LNG ship scheduling and inventory management'. In: European Journal of Operational Research 241.3 (2015), pp. 662–673.

[33] A. Gregory and S. Majumdar. 'Energy Aware Resource Management for MapReduce Jobs with Service Level Agreements in Cloud Data Centers'. In: 2016 IEEE International Conference on Computer and Information Technology (CIT). 2016, pp. 568–577.

[34] R. Grønhaug and M. Christiansen. 'Supply Chain Optimization for the Liquefied Natural Gas Business'. In: Innovations in Distribution Logistics. Ed. by J. A. Nunen, M. G. Speranza and L. Bertazzi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 195–218.

[35] A. M. Ham and E. Cakici. 'Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches'. In: Computers Industrial Engineering 102 (2016), pp. 160–165.

[36] F. Al-Khayyal and S.-J. Hwang. 'Inventory constrained maritime routing and scheduling for multi-commodity liquid bulk, Part I: Applications and model'. In: European Journal of Operational Research 176.1 (2007), pp. 106–130.

[37] P. Laborie. 'An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling'. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Ed. by W.-J. van Hoeve. Cham: Springer International Publishing, 2018, pp. 403–411.

[38] P. Laborie. 'IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems'. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Ed. by W.-J. van Hoeve and J. Hooker. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 148–162.

[39] P. Laborie. Model Presolve, Warmstart and Conflict Refining in CP Optimizer. Aug. 2013.

[40] P. Laborie et al. 'IBM ILOG CP optimizer for scheduling'. In: Constraints 23.2 (2018), pp. 210–250.

[41] P. Laborie and J. Rogerie. 'Reasoning with Conditional Time-Intervals.' In: Jan. 2008, pp. 555–560.

[42] P. Laborie, J. Rogerie, P. Shaw and P. Vilím. 'Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources.' In: Jan. 2009.

[43] D. M. Miller. 'An interactive, computer-aided ship scheduling system'. In: European Journal of Operational Research 32.3 (1987), pp. 363–379.

[44] R. Mohr and G. Masini. 'Good Old Discrete Relaxation'. In: ECAI. 1988.

[45] J. Oppen, A. Løkketangen and J. Desrosiers. 'Solving a rich vehicle routing and inventory problem using column generation'. In: Computers & OR 37 (July 2010), pp. 1308–1317.

[46] D. J. Papageorgiou. 'Optimization in maritime inventory routing'. PhD thesis. Georgia Institute of Technology, 2012.

[47] D. J. Papageorgiou, M.-S. Cheon, S. Harwood, F. Trespalacios and G. L. Nemhauser. 'Recent Progress Using Matheuristics for Strategic Maritime Inventory Routing'. In: (2018). Ed. by C. Konstantopoulos and G. Pantziou, pp. 59–94.

[48] D. J. Papageorgiou, G. L. Nemhauser, J. Sokol, M.-S. Cheon and A. B. Keha. 'MIRPLib – A library of maritime inventory routing problem instances: Survey, core model, and benchmark results'. In: European Journal of Operational Research 235.2 (2014). Maritime Logistics, pp. 350–366.

[49] J. A. Persson and M. Göthe-Lundgren. 'Shipment planning at oil refineries using column generation and valid inequalities'. In: European Journal of Operational Research 163.3 (2005). Supply Chain Management and Advanced Planning, pp. 631–652.

[50] G. Pesant. 'A constraint programming primer'. In: EURO Journal on Computational Optimization 2.3 (2014), pp. 89–97.

[51] D. Popovic, M. Vidovic and G. Radivojević. 'Variable Neighborhood Search heuristic for the Inventory Routing Problem in fuel delivery'. In: Expert Systems with Applications 39 (Dec. 2012), pp. 13390–13398.

[52] J.-F. Puget. 'A Fast Algorithm for the Bound Consistency of Alldiff Constraints'. In: Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence. AAAI '98/IAAI '98. Madison, Wisconsin, USA: American Association for Artificial Intelligence, 1998, pp. 359–366.

[53] R. G. Richey, J. Stacey, M. Natarajarathinam and C. Sox. 'The storage constrained, inbound inventory routing problem'. In: International Journal of Physical Distribution & Logistics Management (2007).

[54] J.-P. Rodrigue, C. Comtois and B. Slack. The geography of transport systems. Jan. 2016, pp. 1–440.

[55] D. Ronen. 'Marine inventory routing: shipments planning'. In: Journal of the Operational Research Society 53.1 (Jan. 2002), pp. 108–114.

[56] N. Roofigari-Esfahan and S. Razavi. 'Uncertainty-Aware Linear Schedule Optimization: A Space-Time Constraint-Satisfaction Approach'. In: Journal of Construction Engineering and Management 143.5 (2017), p. 04016132.

[57] N. Siswanto, D. Essam and R. Sarker. 'Solving the ship inventory routing and scheduling problem with undedicated compartments'. In: Computers and Industrial Engineering 61.2 (2011). Combinatorial Optimization in Industrial Engineering, pp. 289–299.

[58] J.-H. Song and K. Furman. ‘A maritime inventory routing problem: Practical approach’. In: Computers and Operations Research 40.3 (2013). Transport Scheduling, pp. 657–665.

[59] U. N. C. on Trade and Development. Review of Maritime Transport, 2019.

[60] P. Vilím. ‘Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources’. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Ed. by T. Achterberg and J. C. Beck. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 230–245.

[61] P. Vilím, P. Laborie and P. Shaw. ‘Failure-Directed Search for Constraint-Based Scheduling’. In: Integration of AI and OR Techniques in Constraint Programming. Ed. by L. Michel. Cham: Springer International Publishing, 2015, pp. 437–453.