FACULTEIT ECONOMIE EN
BEDRIJFSWETENSCHAPPEN

KU LEUVEN

SCHEDULING LOCKS ON INLAND WATERWAYS

Proefschrift voorgedragen tot
het behalen van de graad van
Doctor in de Toegepaste
Economische Wetenschappen

door

Ward PASSCHYN

NUMMER 532 2016

Doctoral committee

Prof. dr. Frits C.R. Spieksma (advisor)
KU Leuven, Research Centre for Operations Research & Business Statistics

Prof. dr. Dirk Briskorn
Bergische Universität Wuppertal, Chair of Production and Logistics

Prof. dr. Roel Leus
KU Leuven, Research Centre for Operations Research & Business Statistics

Prof. dr. ir. Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC

Prof. dr. ir. Constantinus P.M. van Hoesel
Maastricht University, Department of Quantitative Economics

Prof. dr. ir. Pieter Vansteenwegen
KU Leuven, Centre for Industrial Management / Traffic & Infrastructure

Acknowledgments

It is often said that the journey is more important than the destination. Although obtaining a PhD is undoubtedly a rewarding achievement, it is indeed the four-year journey leading up to it that has turned out to be a truly remarkable experience. Most importantly, pursuing a PhD is not an individual endeavour, and I would like to thank the many guides and fellow-travellers who have contributed to the realization of my fascinating journey as a PhD-student. Since many people should be named here, any attempted list of people to thank is likely to remain incomplete. Let me therefore preface this section by offering my sincerest apologies to anyone who I might have inadvertently forgotten.

First of all, I would like to thank my doctoral advisor Frits Spieksma for the invaluable support and supervision. Frits, I have more than once been amazed by your ability to maintain an excellent balance in many different aspects of research. Among others, the balance between clearly communicating the big picture and unambiguously providing specific results in full detail. The balance between tactfully providing constructive criticism and nevertheless identifying every little mistake and source of confusion. The balance between continuously investigating new questions and alternative methods, and still directing a project towards a coherent conclusion. And perhaps most importantly, the balance between giving me the freedom to set priorities and discover new things, and still setting ambitious deadlines to keep up the pace. Each of the above have contributed to creating a challenging, rewarding, and stimulating research environment. Further, I am also very grateful for the opportunities you have provided me, such as attending many wonderful conferences. It has been a pleasure doing research under your guidance.

Next, I want to express my gratitude towards the members of the doctoral committee: Dirk Briskorn, Roel Leus, Greet Vanden Berghe, Stan van Hoesel, and Pieter Vansteenwegen. Thank you for reading through

the earlier versions of this text and for taking the time to provide me with valuable feedback and suggestions. With your help, the quality and the readability of the text have surely improved a lot. In addition, I also want to thank Roel for making me aware of the PhD opportunities at the Faculty of Economics and Business, and inviting me to apply for a position. I think it is safe to say that this single e-mail, sent four years ago, has had a profound impact on my career and on my life in general.

On the practical side of things, I am very grateful to the Belgian Science Policy Office for providing the funding that made all of this possible, and to KU Leuven for providing me with a nicely located office and many other facilities.

During my time as a PhD-student, I have gotten to know many wonderful colleagues who have all contributed to the fantastic working environment that I was privileged to enjoy. Whether by attending conferences together, enjoying a soup break, setting up group activities after working hours, going for pizza lunches, upholding the cake-day tradition, or by sharing stories of joy and sometimes also frustration, many people have made my stay at the Hogenheuvelcollege unforgettable. A profound thank you to my colleagues and former colleagues of the "fifth floor": Alexander, Annette, Bart S., Bart V., Celien, Daniel, Dries, Fabrice, Fan, Guopeng, Kris, Salim, Sofie, and Vikram, as well as the people from the statistics part of ORSTAT. Another profound thank you goes out to my colleagues and former colleagues of the "fourth floor": Ann, Carla, Dennis, Gert, Hamed, Joeri, Joren, Jorne, Michael, Mieke, Morteza, Patricio, Raïsa, Sarah VdA., Stef, Valeria, and Yannick.

There are also many people from the outside of the academic sphere that I want to thank. At times, it is of course nice to have a distraction from all ongoing occupations and approaching deadlines. For providing this welcome distraction in the form of physical exhaustion while still maintaining an entertaining atmosphere, I want to thank the many people that I have gotten to know at the sports centre's kung fu practices, and in particular my fellow 'old guys' Boris, Sam, and Stefan.

I have many priceless memories of my time in Leuven and owe a great deal of these to the lifelong friends that I have made at the self-proclaimed Willemshome 'residence' during my time as a student. Some of them have taken their first steps in Leuven together with me, almost literally, and may recall a walk discovering what must be one of the most inefficient ways to find a supermarket. Many others have joined our happy bunch over time, and have starred in countless remarkable stories and shared unforgettable experiences.

I also want to thank my parents. You have made me the person that I am today, and I am lucky to have always been able to count on your support and encouragement. Another warm thank you goes out to my brothers and the rest of my family.

Finally, there is one very important person left for me to thank. Sarah, you were always there for me, providing comforting words, encouragement, and certainly showed me more patience than I could have hoped for. Thank you for your love, support, and understanding.

Ward Passchyn
Leuven, June 2016.

Summary

Inland waterways are well-suited for freight transportation due to their reliability and cost-effectiveness. Locks, acting as natural bottlenecks, are present on many inland waterways. We investigate the problem of scheduling such locks from a mathematical point of view. We explore the characteristics of different scheduling problems underlying the operational planning of locks by providing results regarding the computational complexity and by describing models and algorithms for a number of different problem settings.

The first problem setting we consider is that of a single lock consisting of a single chamber. We describe a polynomial-time algorithm which minimizes the total waiting time of ships that need to pass through the lock. This algorithm is then extended to take different practical aspects into account. Next, we focus on a setting featuring multiple locks in series and investigate the computational complexity of the problem of minimizing the total waiting time of ships passing through these locks. Additionally, we introduce and evaluate integer programming models that solve this problem, and show how the speed of ships can be taken into account to model the fuel cost and the related emissions. The models are used to evaluate the trade-off between the objectives of minimizing the travel time and minimizing the emissions. Lastly, we look at a problem setting featuring multiple lock chambers arranged in parallel, and focus on solutions where the chambers can be scheduled such that no ship has to wait before entering the lock. We characterize the existence of such schedules, and describe algorithms for the general setting, as well as dedicated algorithms that solve a number of special cases more efficiently.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

Inland waterways form a natural infrastructure well-suited for the transportation of goods. Since ancient times, rivers have played an important role throughout history, both economically and politically. Though we may never know when the first use of ships occurred in history, evidence (Meijer, 2007) indicates that wooden ships were built for the transport of construction materials in Egypt as early as 2650 BCE. While the first man-made canals served mainly for irrigation, canals built to modify the natural flow of a river appeared, also in Egypt, around 2300 BCE, see e.g. Hattendorf (2007).

In more recent times – even with the emergence of road, railway, and aviation infrastructure – the transport of goods over inland waterways remains an important part of the supply chain and represents a significant portion of the total freight transport. We refer to e.g. Notteboom (2007) for an overview of the recent history of container barge shipping in Europe. In 2003, inland waterway transport respectively represented 31% and 40% of the containerised cargo in the ports of Antwerp and Rotterdam, the two largest European ports by cargo volume.

Locks are needed on many inland waterways. They maintain a water level suitable for navigation while allowing ships to overcome the resulting differences in water level. Locks also provide a way to bypass obstacles such as waterfalls or dams with hydro-power generation facilities. Occasionally, locks may also serve to control the water flow on a river, or act as a barrier for flood protection. As an example, Figure 1.1 shows a lock on a river and a pair of locks near a port.

Due to the time needed to operate these locks – that is, to allow a

(a) (b)

Figure 1.1: Example locks. (a) a tugboat pulling a barge at McAlpine Lock, USA. (© US Army Corps of Engineers.) (b) the Berendrecht and Zandvliet Locks at the Port of Antwerp, Belgium. (© wwuyts, wikimedia commons.)

ship to enter, to change the water level, and to allow the ship to leave on the opposite side – locks constitute a natural bottleneck near ports and along many rivers and canals. In particular, the decisions made with respect to the operating times of one or multiple locks, determine to a large extent the total delay or waiting time incurred by ships travelling through a waterway. Other decisions include the assignment of ships to lock chambers, since multiple lock chambers may be available, or since it may be possible to fit multiple ships in a single lock chamber and to simultaneously serve these ships with a single lockage operation. An overview of the recurring terminology, along with more formal definitions, can be found in Section 1.3.

Throughout this text, different aspects of the lock scheduling problem are investigated. More specifically, our results focus on locks on inland waterways. As will be argued in Section 1.1, these locks differ significantly from locks in ports and, while some aspects of the scheduling overlap, a different solution approach is justified for these distinct settings.

The lock scheduling problem is approached from a theoretical perspective and related to the well-studied field of machine scheduling. In the following chapters, several different problem settings are outlined. Initially, we ignore a number of practical issues which would be present in a real-life scenario. This allows us to investigate the complexity and characteristics

of the 'core' problem, without needlessly complicating the discussion with cumbersome notation or additional specific conditions. The underlying idea is that a good method which tackles the core problem is, in general, easily adapted to take additional practical restrictions into account, or a solution to the core problem could be modified without fundamental changes to obtain a high-quality solution which takes the practical issues into account. The main goal of this text is to improve our understanding of the computational properties of lock scheduling problems, and thereby contribute to enhancing practical decision-making.

The remainder of this chapter is outlined as follows. Section 1.1 describes and motivates the importance of locks and their scheduling by discussing some key statistics. Section 1.2 gives an overview of literature which relates to different aspects of lock scheduling. Additional literature which is specific to one of the later chapters or which provides known results specific to a particular problem setting, will be covered in the chapter where it is relevant. Section 1.3 introduces a number of concepts and gives a formal definition of some terminology which will recur throughout the remaining chapters. Finally, Section 1.4 outlines the following chapters and gives a brief overview of the different problem settings discussed in the remainder of this text, as well as a short summary of the main contributions.

1.1 Context and motivation

In 2014, inland waterway transport accounted for 6.6 % of the total inland freight transport in Europe. Table 1.1 shows the modal split for different EU countries. The European Commission is actively working to promote the use of inland waterways for freight transport as part of the Trans-European Transport Network (TEN-T). More specifically, the NAIADES 'policy package' was adopted in 2006 and subsequently followed up and extended by the NAIADES II package, see e.g European Commission (2012). With this set of policy changes, the European Commission aims to reduce transport-related greenhouse gas emissions by 60 % by 2050. A significant contribution towards this goal will be made by the use of inland waterways, for which the modal share of freight transport is expected to rise by as much as 20 % by the year 2050.

Freight transport over inland waterways contributes to a significant portion of the total logistics chain. The large capacities that can be carried by barge make this a particularly interesting choice for many types of bulk

	Water (%)	Road (%)	Rail (%)
The Netherlands	39	56.1	4.9
Belgium	21.1	64.2	14.7
Romania	19.9	59.4	20.7
Bulgaria	14	76.6	9.5
Germany	12.3	64.4	23.4
Croatia	5.9	76.8	17.3
Austria	4.6	51.7	43.6
France	4.3	80.1	15.6
Hungary	3.7	75.8	20.5
Luxembourg	2.8	95.1	2.1
Slovakia	2.2	76.3	21.5
EU 28	6.6	75.4	18

Table 1.1: Modal split for European inland freight in 2014, for countries with a reported waterway transport share exceeding 1%. Note: shares may not sum up to 100% due to rounding. (Data: Eurostat)

cargo, raw materials, and products with a relatively low value-to-weight ratio. Table 1.2 illustrates this, showing that metals, agricultural cargo, chemicals, and petroleum products constitute over 85 % of the total cargo carried over inland waterways.

The advantages of inland waterway transport include its reliability, cost efficiency, and low pollutant emissions. Additionally, the current traffic on many inland waterways is well below the available capacity, as opposed to the road transport network, for which congestion is an ever-increasing problem in many industrialized countries. The European inland waterway network, however, has the potential for a significant increase in additional traffic, see e.g. European Commission (2013). In a recent report, the European Commission (2015) also promotes a better use of inland waterways in order to relieve heavily congested transport corridors. Promotie binnenvaart Vlaanderen (2013) discusses some statistics for inland waterway transport in Flanders, Belgium. The average load capacity for barges on the Belgian inland waterway network is approximately 1400 tonnes. A barge of this capacity can carry a load of 96 twenty-foot containers, equivalent to approximately 60 trucks, or 45 train wagons. Inland waterway transport is therefore increasingly seen as a mode of transport that can make a significant contribution to sustainable mobility.

Product type	Share (%)
Metal ores and other mining products	23.8
Coke and refined petroleum products	16.4
Products of agriculture	13.5
Coal, lignite, crude petroleum, and natural gas	11.8
Chemicals, chemical products, and nuclear fuel	11.2
Basic metals and fabricated metal products	9.3
Food products, beverages, and tobacco	6.1
Secondary raw materials and wastes	4.1
Other	3.8

Table 1.2: Distribution of goods carried over inland waterways, 2014 average for EU-28 countries. (Data: Eurostat)

Not only is the energy consumption of transport over water approximately 17% of that of road transport and 50% of rail transport, it also has a high degree of safety and its noise and gas emissions are modest.

In the US, total waterborne commerce has risen from about 1700 million tonnes of goods in 1975 up to approximately 2600 million tonnes in 2006; due to the economic crisis it has dropped to a level of about 2200 million tonnes in 2009, and recovered to about 2350 million tonnes in 2014 (US Army Corps of Engineers, 2014). In a report prepared for the State of New York (Goodban Belt LLC, 2010), the potential of the New York Canal System for container-on-barge logistics is extensively described and an increased flow is expected due to Panama Canal expansion. This expansion, which will double the total throughput capacity of the Panama Canal by the installation of additional locks serving larger ships, is scheduled for completion in 2016 (Panama Canal Authority, 2016). In China, 88 million tonnes of freight passed the Three Gorges Dam in 2010; this is nearly 5 times the maximal annual volume reported before 2003 (ChinaDaily, 2011).

Locks are present on many inland waterways as well as near ports. Since port locks need to accommodate large intercontinental trade vessels, the problem of scheduling port locks shows a number of differences when compared to scheduling the smaller inland locks. In the large lock chambers of a port lock, the relative placement of ships within a chamber becomes a lot more involved. Additional safety distances may be required when ships of significantly different sizes are present within a single lock chamber.

Furthermore, since the engines of large container vessels may not be suited for the precise manoeuvring required within a port, these ships must often be accompanied by tug boats which further complicates the mooring of ships within lock chambers. Further, while the main function of a port lock is to ensure that the water level within the port remains unaffected by tidal movements, the tide may still play an important role in the scheduling of these locks. In order to safely navigate the area around the port, some of the largest ships, i.e. those ships which require a large waterway depth for safe navigation, can only leave the port when the tide is high. This results in a so-called 'tidal window' where some of the ships can only leave the port during specific time intervals which must also be reflected in the operating schedule. Finally, the traffic density is typically much higher in port areas than on inland waterways. It is intuitively clear that when the traffic density increases up to the maximum traffic throughput of a lock, the idle time of a lock decreases up to the point where the lock must continually switch back and forth between the upstream and downstream water level, effectively removing the need for any decision-making related to the operating times. It is also for this reason that, for port locks, the scheduling of operating times loses some of its relative importance compared to the ship placement problem. Therefore, we limit the focus of this text to the scheduling of locks on inland waterways.

At several waterway networks, congestion is expected to increase, yielding extra pressure on the locks. Examples are the New York State Canal System (Goodban Belt LLC, 2010) and the North Sea Canal region (van Haastert, 2003). One way to anticipate this increase in traffic density is to expand the size of existing locks, such as for example on the Twente Canal in the Netherlands (Rijkswaterstaat, 2010). Also, new locks are being built; for example, the Kieldrecht lock at the Port of Antwerp (Antwerp Port Authority, 2011), which will become the world's largest lock by volume. Efficient lock schedules can contribute to the attractiveness of waterway transport and help avoid, or maximize the impact of, expensive infrastructural investments.

Our point of view throughout this work is to see the lock as an entity providing a service to the ships. Then, it makes sense to identify a strategy for the lock that optimizes some criterion related to service, such as the total waiting time incurred by ships. Another point of view would be that the lock announces times when ships can enter the lock in order to be transferred, and that the ships simply need to respect these times. Clearly, even in the latter model, there is still a decision to be made concerning these times.

1.2 Related literature

Lock scheduling has not been extensively studied in the scientific literature, although it has recently started attracting an increasing amount of attention. Here, we give a general overview of the existing body of literature related to different aspects of scheduling locks. For additional literature related to the specific problem settings discussed in Chapters 2 to 5, we refer to the respective chapters. There, the relation between existing research and the topics considered in the respective chapters is discussed, as well as how the presented contributions relate to known results from literature. For example, the lock scheduling can be related to the well-studied machine scheduling problem, where locks can be considered to be machines, and ships can be considered to be jobs that require processing on these machines. Chapters 2 and 3 describe the relevant machine scheduling problems and known results in more detail.

An early example of the application of optimization techniques in the context of scheduling locks is the case of the Welland Canal in North America, which allows ships to bypass the Niagara Falls. The St. Lawrence Seaway Authority, that maintains the canal, faced increasing congestion at the locks. Petersen and Taylor (1988) describe an integer programming model for this setting. The authors also discuss a dynamic programming model for scheduling the operations of a single lock, and extend this model to a heuristic that yields an operating schedule for the series of locks along the Welland Canal. A different integer programming model is proposed by Nauss (2008), where all ships are assumed to be present at a lock at a given time, for example due to technical failure, and a sequencing order is needed to clear the queue in a minimal amount of time.

The problem of scheduling a lock consisting of one chamber is treated by Hermans (2014), who presents an algorithm that asserts feasibility with respect to given ship deadlines when the single chamber has unit capacity. Another approach for the single-lock single-chamber setting is described by Smith et al. (2011), who model this problem setting as a two-stage queue and describe a mixed integer programming model as well as a heuristic solution procedure.

Due to the computational effort involved, a majority of works in the literature restrict the attention to simulation models and heuristic solutions. In Smith et al. (2009), for example, simulation models are used in order to aid policy decisions to reduce congestion on the Upper Mississippi River. Different ship sequencing policies for the Mississippi river are also evaluated by Ting and Schonfeld (2001). The importance of an efficient

lock operating strategy is also noted by Caris et al. (2007), who point out that lock scheduling decisions strongly affect the simulated waiting time and suggest efficient decision rules for lock operations as future work.

Results on obtaining optimum solutions to a system of multiple locks as a whole, are more scarce in the literature. The potential of a centralized approach to scheduling has recently attracted more attention in the field. The Dutch waterway management organization Rijkswaterstaat, for example, is shifting its focus from decentralized lock operations towards the fluent operation of certain 'corridors' as a whole, see Kunst (2013). A recent publication by Prandtstetter et al. (2015) introduces a formal definition for the problem of scheduling locks arranged in a sequence, and describes a variable neighbourhood search approach to obtain heuristic solutions.

The research mentioned above concentrates on single-chamber locks. In practice, however, many locks consist of more than one chamber. On the Panama Canal, for example, each lock consists of two identical parallel chambers which can be operated independently. Another example is the Wijnegem lock, situated in Belgium, which connects the Albert Canal to the port of Antwerp. Like all other locks on the Albert Canal, this lock consists of three non-identical chambers. Furthermore, the construction of a fourth lock chamber in Wijnegem is currently under consideration, see also Waterwegen en Zeekanaal NV and nv De Scheepvaart (2014). Ting and Schonfeld (2001) mention a heuristic for a lock consisting of two chambers. A different heuristic yielding approximate solutions for a problem setting with lock chambers arranged in parallel is available online, see Luy (2012). This algorithm has been integrated by Lübbecke et al. (2014) in a heuristic procedure for traffic optimization on the Kiel canal, which connects the Baltic Sea to the North Sea. The only work we are aware of that deals with exact methods for scheduling a lock with multiple chambers is Verstichel et al. (2014b); a mixed integer program is proposed that simultaneously decides upon the packing of ships in chambers and the operating times of the chambers, see also Verstichel (2013). Thus, studying locks with parallel chambers is a practical and largely unexplored problem.

Besides minimizing the total (weighted) flow time, a different objective could be to minimize the fuel consumption for ships passing through the waterway system. While the fuel consumption may be an important economical factor for ship operators, the related emission of greenhouse gases may also be an optimization criterion for governments or waterway organizations. In the context of road transport, a model for the time-

varying vehicle routing problem is proposed by Qian and Eglese (2014), where the goal is to minimize the greenhouse gas emissions.

1.3 Terminology and notation

In this section, we introduce our terminology and notation. First, the fundamental concepts and entities present in different lock scheduling problems are explained. Next, some notation is introduced to ease the description of the different optimization problems. Finally, a graphical representation is given in order to visualize a problem instance and a corresponding feasible solutions.

1.3.1 Concepts

Central in this entire exposition is the *lock*. Since the main concern in this work is to schedule the locks in order to provide a service to ships, the main function of a lock, as far as our discussion is concerned, is to transfer a ship from one side of the lock to the other. We will also refer to the two sides of the lock as *positions*, i.e. a lock has an upstream position (i.e. high water level) and a downstream position (i.e. low water level).

A single lock may consist of multiple *chambers*. For a ship to be processed by a lock, it suffices to pass through one of the lock's chambers. Since each of a lock's chambers can be operated independently, a lock consisting of two chambers is clearly more flexible than a single-chamber lock from an operational point of view. Of course, this introduces additional complexity in the planning process, since the assignment of ships to chambers may drastically influence the performance of a schedule, in particular when the chambers differ in their operating characteristics, described below.

Note that for consistency of terminology, it is assumed that locks are not arranged in parallel. Rather, chambers may be arranged in parallel and constitute a single lock. Note that this does not always reflect the nomenclature observed in practice. For example, the Berendrecht and Zandvliet locks (see also Figure 1.1b) are indeed named 'lock'; under our terminology, however, they would be considered as two independent chambers of a single lock for all intents and purposes. This is strictly a semantic issue with no impact on the underlying scheduling problem.

The act of performing a single operation with one of the lock's chambers is referred to as a *lockage*. That is, a lockage consists of a set of ships

entering a chamber from one side of the lock, upon which the water level inside the chamber changes and the ships can exit the chamber from the other side of the lock. Depending on the direction in which the ships travel, we can immediately distinguish two types of lockages: upwards lockages transfer ships from the downstream side to the upstream side, and downwards lockages transfer ships from the upstream side to the downstream side of a lock.

Note that a lockage may also be empty, i.e. contain no ships, and that such empty lockages cannot always be avoided. Indeed, after serving a ship, the water level in one of the lock's chambers has changed. In order for the same chamber to serve another ship that is waiting at the initial water level, an empty lockage is required to 'recycle' this chamber to its original position; for this reason, such empty lockages are sometimes referred to as 'turnback lockages'.

We define the *lockage duration* of a chamber as the time needed to perform a single lockage with that specific chamber. That is, the lockage duration is the total time required for a chamber to:

1. allow all ships processed by the lockage to enter the chamber,
2. close the chamber doors on the side where the ships entered the chamber,
3. change the water level in the lock to that of the opposite position,
4. open the chamber doors on the side where the ships leave the chamber, and
5. allow all ships to exit the chamber.

In the case of single-chamber locks we will, for simplicity, refer to the lockage duration of the lock, rather than the lockage duration of a chamber. Throughout the following chapters, it will generally be assumed that the lockage duration for a chamber is a known constant, and thus independent of the set of ships that is processed by the chamber. An extension where this is assumption is generalised is discussed in Section 2.5.3.

In addition to the lockage duration, each chamber is also characterized by its *capacity*. While multiple ships may be simultaneously served by a single lockage, the capacity restricts the set of ships within each lockage. This can be achieved in a number of ways. Obviously, each chamber has a limited surface area, which must not be exceeded by the total area of ships present in the chamber at any given time. In some cases,

especially for narrow locks on inland waterways, the width of a chamber can be ignored and the capacity can be expressed as the allowed total length of ships. However, simply comparing the total area of ships to that of the lock ignores a number of important considerations regarding e.g. safety distances and the mooring of ships, as argued in Section 1.1. For locks on inland waterways, these aspects regarding the placement of ships are of lesser importance. Since optimizing the placement of ships is in itself a computationally hard (i.e. strongly NP-complete) problem, see e.g. Verstichel (2013), it is assumed throughout most of the following chapters that the capacity is expressed in terms of a fixed upper bound on the number of ships that may be present within a chamber. This allows us to focus on the complexity of the scheduling problem while still taking a simplified capacity restriction into account.

The above concerns a single lock. In Chapters 3 and 4, a number of problem settings are investigated where multiple locks are arranged in sequence along a river or canal. Also there, our focus remains on the locks, and the waterway can be simplified and modelled as a line. Any restrictions regarding the navigating of ships on this waterway due to the physical layout (e.g. depth, width, curvature, etc.) are ignored. In particular, note that ships travelling at different speeds may overtake each other within such section in between locks, unless otherwise mentioned. In these settings, the locks are assumed to be ordered. Each lock, except for the last lock in the sequence, is then also characterized by the travel distance separating it from the next lock.

There exist many different types of *ships*. As far as the scheduling problem is concerned, we will simply refer to any boat, ship, barge, or any other vessel to be scheduled for passing through a lock, as 'ship'. As far as barges, which may or may not be self-propelled, are concerned, a set of container barges being pushed by a tugboat is thus considered as a single ship. Furthermore, in what follows we consider each ship as a single entity which cannot be split. In contrast, such a fleet of barges could also exceed the length of a lock and be disconnected in smaller groups of barges in order to allow processing by a lock. This practice occurs for example on the Upper Mississippi River, see e.g. Smith et al. (2009), although this is not common on European waterways. With regards to the type of ships to be processed, observe that the set of ships can be chosen to reflect policy decisions with respect to priorities and 'right of way'. For example, smaller recreational vessels may be excluded from consideration if operational policy dictates that freight ships receive priority at all times. In order to further differentiate between ships of different types, a certain priority

value may be assigned to each ship, which is then reflected in the way an objective value is computed.

Two important properties associated with each of the ships are the *arrival time* and *arrival position*. Clearly, in order to assign a ship to a lockage and a corresponding time at which the ship can enter a chamber, the exact time at which the ships arrive should be known, as well as the side of the lock at which each ship arrives. Related to the arrival position (i.e. upstream or downstream) is the *direction* of travel for a ship. We will say that a ship is downstream-travelling if it arrives on the upstream side of a lock, or a sequence of locks, and travels towards the downstream side, and that a ship is upstream-travelling if it arrives on the downstream side and travels towards the upstream side.

Specifying the arrival position suffices in the context of a single lock, and in fact facilitates the notation for additional concepts introduced in Chapter 2. However, referring to the upstream side (or downstream side) in the context of multiple locks in sequence is somewhat ambiguous. Indeed, each of the individual locks then has an upstream side; moreover, the upstream side of one of the locks has the same water level as the downstream side of an adjacent lock. When all ships traverse each of the locks in the sequence, the direction of travel of a ship implicitly specifies where the ship enters and leaves the system. In general, however, this need not be the case. Indeed, consider for example a river with four locks in sequence where a canal introduces a 'side branch' between the second and third lock. There could then exist ships that traverse only the first and second lock, or only the third and fourth lock, in addition to the ships that pass by each of the locks. Alternatively, it may be the case that the destination of a ship is located in between the locks along the waterway. Therefore, the *arrival lock* and *departure lock*, respectively indicating the first and last lock to be visited by a ship, must be explicitly specified. Note that if the arrival and departure lock are not the same, the travel direction for a ship is implied; if a ship only traverses a single lock, however, it remains necessary to specify the travel direction.

In general, ships may be either upstream-travelling or downstream-travelling; this setting will be referred to as the bi-directional setting. As a special case, the so-called uni-directional setting can also be distinguished; in this setting, all ships travel in the same direction.

We focus on the deterministic scheduling problem, i.e. all arrival times and positions are assumed to be known ahead of time. While assuming that future arrival data are known ahead of time may not be entirely realistic for the planning of a long time horizon, two main arguments can be made

in its favour. First, some future arrival information is likely to be available. The guidelines for the River Information Services being implemented in Europe, for example, specify that a ship should notify a lock operator at least two hours before it arrives at the lock. These guidelines are currently enforced for all freight ships on the Rhine and Danube rivers, and will be further extended throughout the European waterway network (Central Commission for Navigation on the Rhine, 2015). Additional estimates of arrival times can also be extrapolated from known arrival times at other locks along the waterway or from GPS data, which is increasingly recorded and broadcast. See, for example, Central Commission for the Navigation of the Rhine (2011) for an overview of the Automatic Identification System, which provides a standard for the automatic collection and communication of ship data, including the current position and heading. Secondly, the time horizon considered may be limited, such as would be the case if a schedule is repeatedly optimized in a 'rolling horizon' procedure in an on-line setting, i.e. a schedule is initially obtained for a limited time horizon with known arrival information, and the obtained solution is re-optimized as more information becomes available over time.

In those settings where the distance between locks becomes relevant, the *ship speed* should also be considered. This speed may either be assumed to be constant throughout the canal for a given ship, or assumed to be variable and adapted to the schedule in such a way that an objective function is optimized. Where the speed is variable, a valid range for the ship speed is specified, i.e. each ship is then characterized by a minimum and maximum speed.

One measure for the performance of a solution is the total waiting time incurred by all ships. We define the *waiting time* of a ship as the time spent by that ship waiting before entering a lock chamber. Note that this excludes the time spent inside lock chambers. For a setting with a single lock, the waiting time can thus be computed as the time between the arrival of the ship and the moment in time where the ship enters a chamber and starts a lockage. In a setting with multiple locks, a ship may incur some waiting time at each of the locks; the total waiting time for a ship can then be defined as the time between the arrival of the ship at its arrival lock and the time at which it leaves its departure lock, minus the total lockage duration of all lock chambers by which it was served and minus the total travel time. Note that in a problem setting with only single-chamber locks, the total time spent inside lock chambers is necessarily constant, so that the objective of minimizing the total waiting time is equivalent to the commonly used objective of minimizing the total

completion time in a machine scheduling context.

The different problems we consider are concerned with finding 'high-quality' schedules. A *schedule* consists of an assignment of ships to lockages for each chamber, coupled with specifying the starting times of these lockages. In a setting with parallel chambers, this also includes an assignment of ships to chambers. We say that a schedule is a *feasible schedule* if:

1. each ship is processed by each lock that it is required to pass, i.e. by one of the chambers of each of these locks,

2. the order in which a ship passes the locks is consistent with its direction of travel,

3. the time at which a ship enters a lock is consistent with the ship's arrival time and with the travel time and starting time of a previous lockage containing that ship,

4. for each lock chamber, the starting times of its lockages are such that these lockages do not overlap,

5. lockages alternate between upwards and downwards lockages, and ships are only processed by the type of lockage that corresponds to their direction of travel,

6. the number of ships in any given lockage does not exceed the chamber capacity.

The quality of a feasible schedule can be determined in a number of ways. One obvious choice is to consider the waiting time of ships and to aim for a solution that has minimum total waiting time. This objective function is discussed in Chapters 2 to 4. Additionally, Chapter 4 considers the emissions, which can be related to the speed of ships. Another approach is to look for any feasible schedule. In Chapter 5, a schedule is said to be feasible if no ship incurs any waiting time, and the objective reduces to finding any feasible solution.

1.3.2 Notation

In order to provide formal proofs in the following chapters, we introduce some notation for the concepts outlined above. Here, an overview is provided for the notation which is common to all following chapters. Additional notation for more specific concepts will be defined where needed.

For clarity, lowercase symbols will be reserved for decision variables. All input data, i.e. parameters, are denoted in uppercase. Sets are denoted in calligraphic script. Some exceptions will be made to this notation style for frequently used symbols that have been adopted by convention. One notable exception is the use of parameters n and m, which are typically used to reflect the instance size; in our case, they are used to represent the number of ships and the number of locks in an instance respectively. Another exception is the use of V and E for a set of vertices and a set of edges in a graph, respectively. Table 1.3 provides an overview of the notation introduced here; we briefly discuss each introduced symbol below.

Central in each problem instance are a set of L locks $\mathcal{L} = \{1, \ldots, L\}$, and a set of S ships $\mathcal{S} = \{1, \ldots, S\}$. Since the locks must be visited in an order corresponding to a ship's direction of travel, a total order is assumed on $\mathcal{L}$ so that the locks are arranged in the order in which they are visited by a ship that traverses the entire waterway in the upstream direction. Equivalently, this order could be reversed and correspond to a ship travelling in the downstream direction. For the set of ships, a total order is generally assumed to be given so that ships are sorted by their time of arrival; additional details on the sorting of ships are covered in the later chapters. We also define the sets $\mathcal{U}$ and $\mathcal{D}$, containing all upstream-travelling ships and all downstream-travelling ships respectively. Observe that, thus, $\mathcal{U} \cup \mathcal{D} = \mathcal{S}$, and $\mathcal{U} \cap \mathcal{D} = \emptyset$.

For each ship $s \in \mathcal{S}$, the arrival time is denoted by A_s. Note that the travel direction of a ship s is easily determined by checking whether $s \in \mathcal{U}$ or $s \in \mathcal{D}$. The arrival lock and departure lock of ship s are respectively denoted by L_s^A and L_s^D. Additionally, for ease of notation when discussing single locks, it will be helpful to define the arrival position of a ship s as P_s. Finally, each ship also has a fixed minimum and maximum travel speed, represented by $V_s^{\min}$ and $V_s^{\max}$ respectively. In a setting where the speed is fixed, we have $V_s^{\min} = V_s^{\max}$.

For a given lock $l \in \mathcal{L}$, let $\mathcal{C}_l$ be the set of chambers in l. Each chamber $c \in \mathcal{C}_l$ has a lockage duration $T_{l,c}$ and a capacity $C_{l,c}$. Whenever a setting is considered where each lock consists of a single chamber, this notation is simplified to T_l and C_l. Where multiple locks are arranged in a sequence, the length of the waterway section between locks l and $l+1$ will be denoted with S_l.

$\mathcal{S}$	$= \{1, \ldots, S\}$	the set of all arriving ships,
$\mathcal{U}$		the set of upstream-travelling ships,
$\mathcal{D}$		the set of downstream-travelling ships,
$\mathcal{L}$	$= \{1, \ldots, L\}$	the set of all locks,
$\mathcal{C}_l$	(for $l \in \mathcal{L}$)	the set of chambers in lock l,
A_s	(for $s \in S$)	the arrival time of ship s,
L_s^A	(for $s \in \mathcal{S}$)	the arrival lock of ship s,
L_s^D	(for $s \in \mathcal{S}$)	the departure lock of ship s,
$V_s^{\min}$	(for $s \in \mathcal{S}$)	the minimum speed attainable by ship s,
$V_s^{\max}$	(for $s \in \mathcal{S}$)	the maximum speed attainable by ship s,
$T_{l,c}$	(for $l \in \mathcal{L}, c \in \mathcal{C}_l$)	the lockage duration for chamber c of lock l,
$C_{l,c}$	(for $l \in \mathcal{L}, c \in \mathcal{C}_l$)	the lock capacity for chamber c of lock l,
S_l	(for $l \in \mathcal{L} \setminus \{L\}$)	the distance between locks l and $l+1$.

Table 1.3: Summary of notation

1.3.3 Graphical representation of instances

A frequently used method for visualising schedules in transportation is the time-distance diagram. These diagrams are often used to track the movement of vehicles such as trains. For an example in the context of inland waterways, see e.g. Lübbecke et al. (2014) for an application to the Kiel canal.

This method can be easily extended to include locks. An example visualisation for three locks in sequence is shown in Figure 1.2. In the figure, time passes from left to right and the vertical axis denotes the position of the ship in the waterway. The arrival of each ship, over time, is marked with an 'X'. Each tilted line corresponds to a lockage by one of the lock's chambers. Recall the constraints that require that lockages for any given lock chamber must not overlap and must alternate between upwards and downwards lockages. Visually, this corresponds to the requirement that the tilted lines only intersect at the lines corresponding to the water levels of a lock and that, for each lock, lines with an upward slope alternate with lines with a downward slope. This makes it easy to verify these constraints visually.

In the figure, the tilted lines that represent lockages are annotated with the ships contained in the lockage. Note that, strictly speaking, this assignment of ships to lockages should be identified in order to uniquely determine the resulting schedule. However, it is generally acceptable to assume that a ship will not 'needlessly wait' and thus enters the first

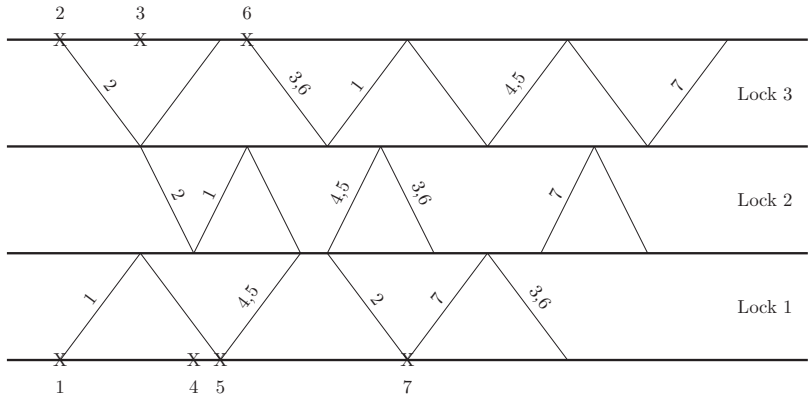


Figure 1.2: Visualisation for a system of three locks in sequence. Each lock consists of a single chamber with a capacity of 2.

available lockage corresponding to its direction of travel. If the capacity of a lock chamber would be exceeded, the ships in a chamber may be selected on a first-come first-served basis, unless otherwise mentioned.

Note that, for multiple locks in a sequence, the distance between two locks is not represented in Figure 1.2. In the figure, the travel time is assumed to be such that once a ship is in a given position, it may enter the next lockage in its direction of travel. Obviously, this need not hold in general. An alternative visualisation that does represent the travel time visually is shown in Figure 1.3. Here, a horizontal line no longer exclusively corresponds to a water level; rather, the vertical coordinate represents the distance from an arbitrary origin, while the locks remain represented by a vertical segment between the lines corresponding to its upstream and downstream position. In Figure 1.3, observe that ship 4 overtakes ship 2 between the first and second lock.

1.4 Outline

We conclude this chapter with a brief overview of the problem settings discussed in each of the following chapters, and a summary of the main results.

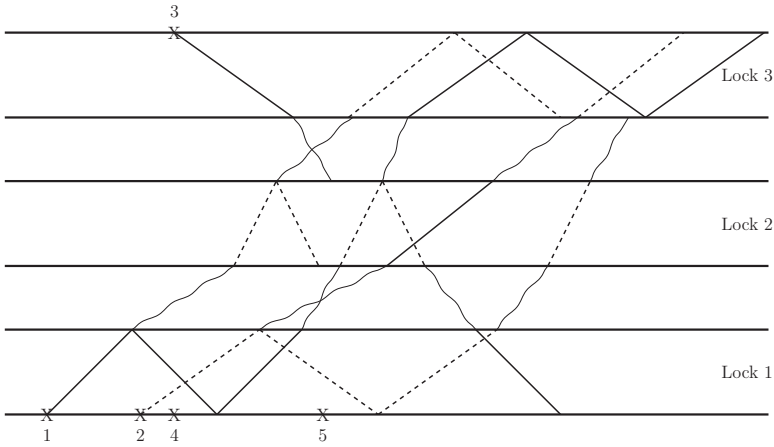


Figure 1.3: Visualisation for three locks. Each lock consists of two chambers with unit capacity. Travel time is indicated by waved lines. Notice that overtaking is allowed, i.e. in contrast to the lines representing lockages, waved lines are allowed to intersect.

Scheduling a single lock Chapter 2 covers the scheduling of a single lock consisting of a single chamber. Given a single lock with known lockage duration and known capacity, and given a set of ships, each characterized by an arrival time and a direction of travel, the problem consists of finding a solution that minimizes the total waiting time of the ships. A dynamic programming algorithm is proposed that solves this so-called lockmaster's problem in polynomial time. We further extend the algorithm so that it can be applied in more realistic settings, taking into account capacity, ship-dependent handling times, ship priorities, and water usage. In addition, we compare the performance of this new exact algorithm with the performance of some (straightforward) heuristics in a computational study.

The computational complexity of scheduling locks in sequence Chapter 3 considers a problem setting where multiple locks are arranged along a line, as is frequently the case on a river or canal. The focus remains on minimizing the total waiting time. We investigate the computational complexity of this problem. More specifically, we show that minimizing the total waiting time is strongly NP-hard, even for two identical locks and with all ships travelling in the same direction. A second NP-hardness

result can be obtained for a related setting with identical ships that are allowed to travel in either direction. Additionally, we introduce a class of so-called synchronised schedules, and describe two problem settings where the existence of an optimum solution which is synchronised can be guaranteed. The claim that bi-directional travel contributes fundamentally to the computational complexity of this problem is further reinforced by describing a polynomial time procedure for a uni-directional setting with identical locks and identical ships travelling in the same direction.

Mathematical programming for locks in sequence and emissions Chapter 4 continues the setting with locks arranged in a sequence. Two distinct mathematical programming models are introduced and their performance is compared empirically. The quality of solutions obtained by solving these models is then compared to that of a heuristic based on an iterative single-lock solution method, showing that significant waiting time reductions can be achieved by scheduling the sequence of locks in its entirety, as opposed to decentralized solution methods. Furthermore, it is shown how the models can be extended to include the ship speed as decision variables in order to optimize the fuel consumption and closely related pollutant emissions. Since the objectives of minimizing waiting time and minimizing total emissions are conflicting, the trade-off between these two objective functions is investigated through a computational study for problem instances based on real-life arrival data.

No-wait scheduling for locks with parallel chambers Chapter 5 discusses a single lock with multiple chambers. Since the chambers may differ in terms of lockage duration or capacity, the assignment of ships to chambers is an important aspect of the scheduling for the lock. We focus on schedules where no ship incurs any waiting time. We show how this problem relates to known interval scheduling problems, as well as to a particular graph colouring problem on multiple unit interval graphs. We explore the relationships between these problems and discuss the complexity of different problem variants. In particular, for a lock consisting of two chambers we are able to characterize the feasible instances and use this result to obtain an efficient solution algorithm. We also provide an efficient algorithm for the special case with identical lock chambers. Furthermore, we describe a dynamic programming approach for the more general case with arbitrary chambers, and prove that the problem is strongly NP-complete when the number of chambers is part of the input.

Chapter 2

Scheduling a single-chamber lock[1]

This chapter concerns the scheduling of a single lock that consists of a single chamber. Section 2.1 starts with a formal definition of a basic problem setting that serves as a reference throughout this chapter. Section 2.2 continues with an overview of literature relating to this basic problem setting; our setting is very similar to the scheduling of a batch processing machine.

The contributions presented in this chapter can be summarized as follows. We show that (i) our basic single-lock single-chamber scheduling problem can be solved by means of a dynamic programming algorithm that runs in $O(n^3)$ time (Section 2.3), (ii) a speed-up up to $O(n^2)$ time can be achieved for this algorithm in the basic setting (Section 2.4), (iii) the algorithm can be extended to solve problem variants with capacities, ship-dependent handling times, ship priorities, non-uniform lockage times, or settings with a limited number of lockages (Section 2.5). Additionally, we investigate the performance of several heuristics by running them on randomly generated instances that possess real-life characteristics (Section 2.7).

[1]The research presented in this chapter, formatted as a journal article, has been published in the European Journal of Operational Research, see Passchyn et al. (2016c) for the article version. (doi: 10.1016/j.ejor.2015.12.007)

2.1 Problem definition

We first discuss a basic setting which acts as our core problem throughout this chapter: the lockmaster's problem. The results obtained for this problem will serve as a basis for dealing with more realistic settings later on.

Consider a lock consisting of a single chamber. As described in Section 1.3, let $\mathcal{S}$ represent the set of ships that arrive at this lock over time, and let $|\mathcal{S}| = n$. For each ship $s \in \mathcal{S}$, let A_s represent its time of arrival, and P_s the position where it arrives. Let $P_s = 1$ if ship s arrives on the upstream side of the lock, and $P_s = 0$ if s arrives on the downstream side. For convenience, we also assume that a total order $1 < 2 < \ldots < n$ is imposed on $\mathcal{S}$ so that ships are ordered by non-decreasing arrival time, i.e. $A_i \leq A_{i+1}$ for $i = 1, \ldots, n-1$. Note that multiple ships may have the same arrival time; the total order thus breaks these 'ties' arbitrarily. It is important to emphasize that we do not require ships to enter the lock in the order imposed on $\mathcal{S}$, except in the cases (see Sections 2.5.1 and 2.5.3) where we explicitly state this as a requirement.

Let T denote the *lockage duration* of the lock's chamber. We may assume $T > 0$ as the problem becomes trivial for $T = 0$. We further assume that all data are integral. Our goal is to find a feasible schedule that minimizes the total waiting time of all ships. Note that we do not consider the chamber's capacity in this basic setting, any set of ships that travel in the same direction can be processed together by a single lock, i.e. the chamber capacity C is assumed to be infinite.

This particular problem, to which we refer as the lockmaster's problem, is a simplified version of reality. However, we see this problem as a basic problem underlying any practical lock scheduling problem.

2.2 Relation to literature

We mention some results known from literature which relate to the lockmaster's problem and describe how results for the lockmaster's problem extends the existing research on machine scheduling problems.

Nauss (2008) discusses a mathematical programming model to minimize the time required to clear a queue at a lock, for example due to a technical outage. Verstichel et al. (2014b) describe a model for a general lock scheduling problem, which includes the placement of ships within lock chambers. An alternative mathematical programming model for a

generalised setting is also discussed in Chapter 4.

The main disadvantage to (mixed) integer programming models is the rapid increase of the required computation time as the instance increases in size. Below, we discuss related works which achieve exact solutions in polynomial time. Hermans (2014) considers a unit-capacity special case of the lockmaster's problem, i.e. the chamber capacity is assumed to be $C = 1$. An $O(n^4 \log n)$ algorithm is described that asserts whether each of the ships can be processed so that it arrives before a pre-defined deadline. The unit-capacity setting where total waiting time is minimized is covered by Petersen and Taylor (1988), who present an $O(n^2)$ dynamic programming algorithm. Our results in Section 2.4 show that a similar result holds when the lock capacity is infinite. Furthermore, we show in Section 2.5.1 that the problem can be solved in $O(n^4)$ time when the lock has arbitrary capacity C.

A number of related results can also be found in the well-studied field of machine scheduling. The scheduling of a machine which allows batch processing, in particular, corresponds to a chamber which processes multiple ships simultaneously. Smith et al. (2011) relate traffic operations at a river lock with a variant of the job shop scheduling problem with sequence dependent setup times and a two-stage queuing process. The lockmaster's problem is more general in two ways: ships, i.e. jobs, have release dates, and multiple ships can be processed together in a single lockage.

Suppose that the lockmaster's problem only has downstream-travelling ships, i.e. consider the uni-directional case. The lock can be seen as a batching machine; the jobs are the arriving ships with release dates and equal processing times, and the flow time of a job corresponds to the waiting time of a ship. Following the notation of Baptiste (2000) this is problem $1 \,|\, \text{p-batch},\ b = n,\ r_j,\ p_j = p \,|\, \sum F_j$. In words: the problem has a single parallel batching machine with unrestricted capacity ($b = n$), release dates on the jobs, and uniform processing times. The objective is to minimize the sum of flow times ($\sum F_j$). Baptiste (2000) shows that this problem is polynomially solvable for a variety of objective functions. Cheng et al. (2005) developed an $O(n^3)$ algorithm for $1 \,|\, \text{p-batch},\ b = n,\ r_j,\ p_j = p \,|\, f$ where f can be any regular objective function. Condotta et al. (2010) show that feasibility of the same problem with bounded capacity and deadlines can be checked in $O(n^2)$, even for a setting with parallel batching machines. Ng et al. (2003) study a single machine serial batching scheduling problem with release dates and identical processing times. Machine setup only happens after arrival of the final job in a batch

and there is a fixed setup time equal to s; the completion time of a batch is equal to the sum of the processing times of the jobs in the batch. An $O(n^5)$ algorithm is presented for this problem setting. By choosing $s = T$ and $p_j = 0$ for all $j \in \mathcal{S}$, it can be seen that the uni-directional lockmaster's problem is a special case of this batching machine scheduling problem.

The lockmaster's problem with bi-directional travel clearly generalizes the uni-directional setting which corresponds to these machine scheduling problems. Indeed, there are then two families of jobs, and only jobs of the same family can be together in a batch. A requirement is that, in our case, processing a batch of one family needs to be alternated by processing a batch corresponding to jobs of the other family, although a batch can possibly be empty. That is, it is not possible to process two batches of the same family consecutively.

The concept of a "family" of jobs is also described by Webster and Baker (1995), be it without a batch processing machine. They deal with a scheduling problem in which setup times can be reduced by consecutively scheduling jobs of the same family. This problem setting is also known as batch scheduling with job compatibilities. Jobs within a batch need to be pairwise compatible, and these compatibilities can be expressed using a compatibility graph. Boudhar (2003) and Finke et al. (2008) study different variants of these batch scheduling problems when the compatibility graph is bipartite or an interval graph. The compatibility graph of the lockmaster's problem is the union of two cliques.

The lockmaster's problem can be summarized as $1 \,|\, \text{s-batch}, b = n, r_j, \Phi = 2, s_{fg}, p_j = 0 \,|\, \sum F_j$, with $s_{fg} = 2T$ if $f = g$ and $s_{fg} = T$ if $f \neq g$, where Φ refers to the number of families and s_{fg} to the setup times between batches. For a review on scheduling a batching machine we refer the reader to Potts and Kovalyov (2000) and Brucker et al. (1998). Another related problem is studied by Lee et al. (1992) who develop dynamic programming algorithms for scheduling a batching machine with release dates, deadlines, and constant processing times where the goal is to minimize makespan or to minimize the number of tardy jobs. In conclusion, the complexity of the lockmaster's problem does not immediately follow from results in the machine scheduling literature.

2.3 A solution for the lockmaster's problem

We construct a graph such that the shortest path in the graph corresponds to an optimal schedule for the lockmaster's problem. We show that the

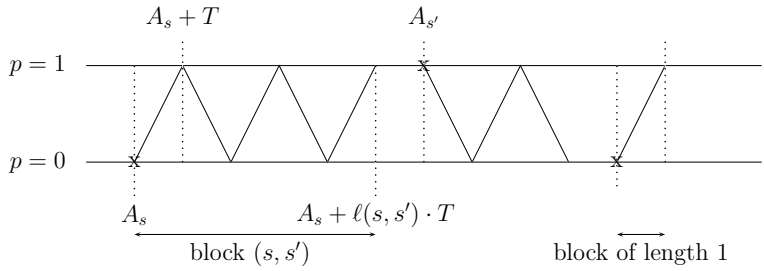


Figure 2.1: Lockmaster's problem: concepts

corresponding shortest path algorithm runs in $O(n^3)$ time. In addition, we propose a more sophisticated implementation in Section 2.4, which results in an $O(n^2)$ algorithm.

Recall that a solution to the lockmaster's problem, i.e. a schedule, consists of an assignment of the ships to a set of lockages, each starting at some moment in time. However, some schedules are more interesting than others. To describe these schedules, we first introduce some additional definitions: let a pair of lockages be called *consecutive* when the starting time of the second lockage equals the starting time of the first lockage plus T. Clearly, after two consecutive lockages starting at time t, the lock is back at the same position at time $t + 2T$ as it was at time t. A set of lockages $\{1, \ldots, \ell\}$ is called consecutive when the pairs of lockages $(i, i+1)$ are consecutive for all $1 \leq i < \ell$. Clearly, when a set of ℓ consecutive lockages starts at time t, it finishes at $t + \ell T$. Recall that the arrival time and position of a ship s are denoted by A_s and P_s respectively. We now define blocks of consecutive lockages starting at the arrival of a ship s and ending before the arrival of a ship s':

Definition 2.1. *For each pair $s, s' \in \mathcal{S}$ with $P_s = P_{s'}$ and $A_s + 2T < A_{s'}$, we define a block $B = (s, s')$ as the set of $\ell(B)$ consecutive lockages that starts at A_s and ends at $A_s + \ell(B)T$, where $\ell(B)$ is the largest even integer such that $A_s + \ell(B)T < A_{s'}$.*

For each pair $s, s' \in \mathcal{S}$ with $P_s \neq P_{s'}$ and $A_s + T < A_{s'}$, we define a block $B = (s, s')$ as the set of $\ell(B)$ consecutive lockages that starts at A_s and ends at $A_s + \ell(B)T$, where $\ell(B)$ is the largest odd integer such that $A_s + \ell(B)T < A_{s'}$.

Figure 2.1 illustrates these definitions. In the figure, observe that $\ell(s, s') = 5$. We can now describe a class of solutions that exhibits a specific structure.

Definition 2.2. *A schedule is called a block-schedule if it consists of a sequence of blocks $B_1, \ldots, B_k$ with $B_i = (a_i, b_i)$, where $b_i = a_{i+1}$ for each i with $1 \leq i < k$, and with a final set of consecutive lockages that starts at A_{b_k} and ends not later than $A_n + 3T$.*

Notice that the distinguishing feature of a block-schedule is that each lockage starts at an arrival time or it directly follows the previous lockage. Observe that the following properties characterize block-schedules:

Property 2.1. *Each lockage either directly follows upon a previous lockage, or starts upon some A_s while containing ship s.*

Property 2.2. *The length of a period in which there is no lockage, a so-called* idle *period, is at most $2T$.*

Property 2.3. *The final lockage does not end later than $t(n) + 3T$.*

We can then state the following result.

Lemma 2.1. *There is an optimum schedule that is a block-schedule.*

Proof. It is easily verified that any schedule which satisfies Property 2.1-Property 2.3 is a block-schedule, and that any block-schedule satisfies these properties.

We show that any schedule not satisfying the above properties can be transformed into a schedule satisfying them, without increasing the objective value. Consider some schedule that does not satisfy Property 2.1. Let t denote the earliest time where a lockage starts such that t is neither the ending time of a previous lockage nor the arrival time A_s of some ship $s \in \mathcal{S}$ contained in this lockage. Since at $t - \epsilon$ with $\epsilon > 0$ and small, the lock is idle, and since no ships contained in the lockage arrive between $t - \epsilon$ and t, we can start this lockage at time $t - \epsilon$ without increasing the objective value. This argument is easily repeated until the lockage starts at either (i) the latest arrival time of a ship contained in it, or (ii) the ending time of the preceding lockage.

Next, consider a solution where no lockage occurs during an idle period with a length no less than $2T$. At the beginning of this period, two additional lockages can be scheduled, reducing the length of the idle period

by $2T$ time units. By repeating this step we end up only with empty periods shorter than $2T$. (Property 2.2).

Further, assume that a solution exists with a final lockage that does not end before $A_n + 3T$. Then, the last lockage contains no ships and, hence, can be dropped. By repeating, we obtain a solution which ends not later than $A_n + 3T$ (Property 2.3).

Now consider an arbitrary optimum schedule. It immediately follows that, by applying the above, we obtain an optimum block-schedule. □

Furthermore, since all ships are identical, ships can be interchanged in any solution to the lockmaster's problem, such that the following property holds:

Property 2.4. *[FCFS] For each pair of ships $s, s' \in \mathcal{S}$ with $P_s = P_{s'}$ it holds that, if $s < s'$, then ship s will leave the lock not later than ship s'.*

Now, we describe an algorithm that solves the lockmaster's problem in polynomial time by finding an optimum block schedule. The basic idea is to build a directed acyclic graph $G = (V, E)$ with a given cost c_e for each $e \in E$. The arcs in the graph represent either (i) the idle time before the first block, (ii) a block in a schedule, or (iii) the final set of consecutive lockages until all ships have been served. We first describe the construction of the nodes and arcs in the graph, and specify the cost on each of the arcs. We then argue that a path in this graph with a certain cost corresponds to a block-schedule with a total waiting time equal to this cost, and vice versa. Thus, a shortest path corresponds to an optimum solution to the lockmaster's problem.

2.3.1 Constructing the graph

There is a node σ, a node τ, and for each ship $s \in \mathcal{S}$, there is a set of $n+2$ nodes that we denote by a *layer* of nodes $L(s)$. One node from this layer is called the *top node*, indicated by s_{top}. All other nodes of the layer are indexed by $k = 0, \ldots, n$, and are denoted by s_k. Hence the node-set V has $O(n^2)$ nodes in total.

There are five types of arcs, namely arcs leaving σ, arcs entering τ, so-called block1 arcs, block2 arcs, and 0-cost arcs. We now describe these arcs and the corresponding costs; Figure 2.3 illustrates the definitions that follow graphically, for the example instance shown in Figure 2.2, assuming a lockage time of $T = 30$. Note that the notation P_s allows us, for a single lock, to conveniently refer to the position opposite to the arrival of

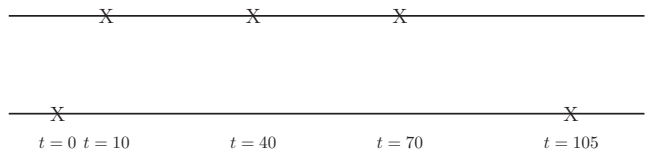


Figure 2.2: Lockmaster's problem: example instance

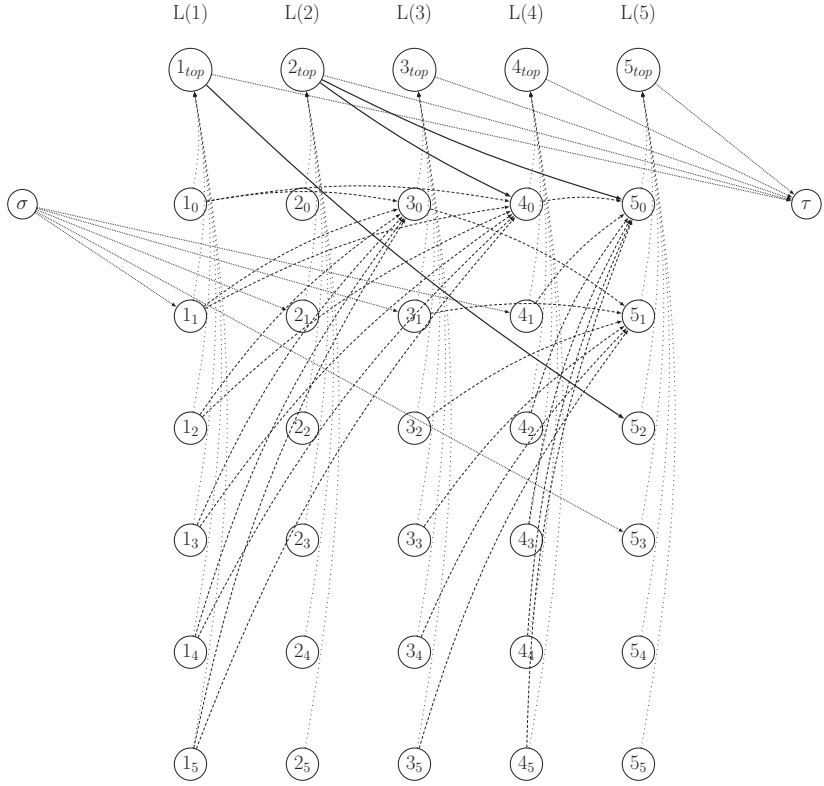


Figure 2.3: Lockmaster's problem: graph corresponding to the example instance of Figure 2.2

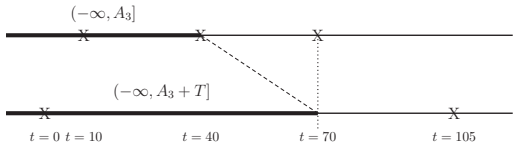


Figure 2.4: Waiting time for the arc $(\sigma, 3_1)$. The highlighted intervals contribute to the cost on the arc; the dashed line represents the first lockage of the next block, starting with ship 3.

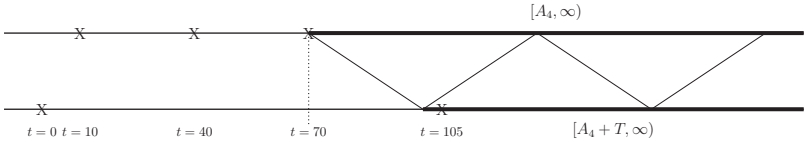


Figure 2.5: Waiting time for the arc $(4_{\text{top}}, \tau)$. The lockages shown serve all remaining ships, starting at the arrival of ship 4.

ship s as position $1 - P_s$, which simplifies the formal description of the construction below.

There is an arc from σ to a single node from each layer $L(s)$ with $s \in \mathcal{S}$, say node s_k with $0 \leq k \leq n$. The value of k is determined by the number of ships arriving in $(-\infty, A_s + T]$ at position $1 - P_s$. This arc represents the situation where the first block of a schedule starts at time A_s and position P_s. The cost of this arc equals the waiting time accumulated at time A_s of all ships arriving in $(-\infty, A_s]$ at position P_s, plus the waiting time accumulated at time $A_s + T$ of all ships arriving in $(-\infty, A_s + T]$ at position $1 - P_s$. For example, the arc from σ to layer $L(3)$ in the example instance ends at node 3_1. Note that there is only one ship that arrives in the interval $(-\infty, 70]$ at the position $1 - P_3 = 0$. The intervals contributing to the waiting time reflected in the cost of the arc are illustrated in Figure 2.4.

There is an arc from a top node s_{top} from each layer $L(s)$ with $s \in \mathcal{S}$ to node τ. The cost of this arc equals the waiting time of all ships arriving in the interval $(A_s, A_n]$ at position P_s, and in $(A_s + T, A_n]$ at position $1 - P_s$ in a solution where a set of consecutive lockages serves all remaining ships, starting at the arrival of ship s. An illustration for the waiting time contributing to the cost of example arc $(4_{\text{top}}, \tau)$ is provided in Figure 2.5.

We now describe the block2 arcs; these arcs correspond to blocks with

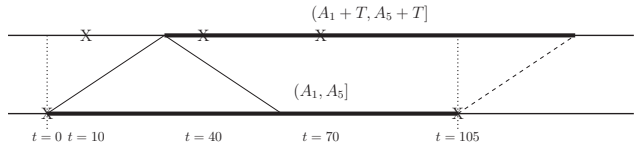


Figure 2.6: Waiting time for the arc $(1_{\text{top}}, 5_2)$. At the end of the block $(1, 5)$, consisting of the two consecutive lockages, two ships are waiting opposite to the arrival position of ship 5; the dashed line represents the first lockage of the next block, starting at ship 5.

a length of at least two lockages. For each block $B = (a, b) \in \mathcal{S} \times \mathcal{S}$ with $\ell(B) \geq 2$, there is an arc from node a_{top} to node $b_k \in L(b)$, where k equals the number of ships that arrive at position $1 - P_b$ in the interval $(A_a + (\ell(B) - 1)T, A_b + T]$. That is, k equals the number of ships waiting in the position opposite to the arrival position of ship b if the next block starts with ship b. The cost of such an arc equals the waiting time accumulated in block B at time $A_b + T$ for, in case $P_a = P_b$, all ships arriving in $(A_a, A_b]$ at position P_a, and in $(A_a + T, A_b + T]$ at position $1 - P_a$ or, in case $P_a \neq P_b$, all ships arriving in $(A_a, A_b + T]$ at position P_a, and in $(A_a + T, A_b]$ at position $1 - P_a$. An example is illustrated in Figure 2.6 for the block $(1, 5)$.

We now describe the block1 arcs; these arcs correspond to blocks consisting of a single lockage. For each block $B = (a, b) \in \mathcal{S} \times \mathcal{S}$ with $\ell(B) = 1$, there is an arc from each node a_k for $0 \leq k \leq n$ to some node b_l from layer $L(b)$ where l equals the number of ships that arrive at position $1 - P_b$ in the interval $(A_a, A_b + T]$. The cost of such an arc consists of two parts, first, the waiting time accumulated at time $A_b + T$ of all ships arriving in $(A_a, A_b + T]$ at position P_a, and in $(A_a + T, A_b]$ at position P_b; and second, $k \times (A_b - A_a - T)$. Figure 2.7 illustrates the arc $(1_1, 3_0)$ from the example instance, corresponding to a block of length 1. Notice that all ships that were already waiting in position P_b before time $A_a + T$ are served at time A_b, while their waiting time reflected in the cost of earlier blocks is counted up to $A_a + T$. Since $A_b > A_a + T$, we need to take this additional waiting time into account, which is achieved by the second term described above. Note that, by construction, the index k of a_k corresponds precisely to the number of such waiting ships.

Finally, within each layer $L(s)$, with $s \in \mathcal{S}$, there is an arc with cost 0 that goes from each node s_k, with $k = 0, \ldots, n$, to node s_{top}; these are

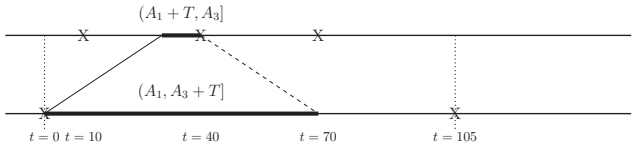


Figure 2.7: Waiting time for the arc $(1_1, 3_0)$. At the end of the block $(1, 3)$, consisting of a single lockage, no ships are waiting opposite at position $1 - P_3$; the dashed line represents the first lockage of the next block, starting at ship 3.

the 0-cost arcs. This completes the description of graph G. We now prove a lemma that establishes the correspondence between block-schedules and paths in G.

Lemma 2.2. *A σ-τ path in G corresponds to a block schedule and vice versa.*

Proof. Consider some path from σ to τ in G. Clearly, the path will visit some nodes of some layers. More precisely, the path can only visit a node in some layer by entering a node s_k in $L(s)$ for some $k \in \{0, \ldots, n\}$. The cost defined above, of any arc entering node s_k assumes that the lock will move at time A_s. Next, there are two ways of leaving node s_k: either, the path proceeds with a 0-cost arc to the top node, or it proceeds with a block1 arc. Proceeding with a 0-cost arc implies that the lock will move at time $A_s + T$, since the path then visits a top node and since all arcs leaving a top node correspond to blocks of length at least 2. Proceeding with a block1 arc to some node b_l means that there is no lockage starting at $A_s + T$. In this situation, by construction, k ships are waiting at position $1 - P_s$ at time $A_s + T$. The waiting time of these ships after $A_s + T$ is included in the cost of the block1 arc leaving node s_k. Hence, the cost on the arc entering node s_k is defined appropriately. Finally, note that the arcs leaving σ represent the waiting time before the first block and the arcs entering τ represent the final set of consecutive lockages. It follows that the cost of each path from σ to τ corresponds to the total waiting time of a block schedule. The reverse is easy to see: any block schedule can be mimicked by choosing the appropriate arcs. □

Theorem 2.1. *The lockmaster's problem can be solved in $O(n^3)$ by a straightforward implementation of a shortest path algorithm on the acyclic graph G.*

Proof. The previous lemma, combined with Lemma 2.1 and the observation that the graph G is acyclic and contains $O(n^3)$ arcs imply this result. □

Notice that the problem definition does not impose a starting position for the lock. In case the starting position of the lock is pre-specified, it is easy to modify the algorithm to deal with this feature.

2.4 A faster algorithm for the basic problem

We describe here a more efficient implementation that uses the structure in the graph G to solve the lockmaster's problem. The resulting complexity is $O(n^2)$. The method used to obtain this speed-up shows similarities to the approaches proposed by Wagelmans et al. (1992) and Federgruen and Tzur (1991), who independently describe a procedure to solve the well-known lot sizing problem of Wagner and Within in linear time. A similar result was also independently obtained by Aggarwal and Park (1993).

First, we define $sp(a_k)$ as the shortest path length from σ to node a_k, with $0 \leq k \leq n$. Further, define $sp^{a_k}(b_i)$ as the shortest path length to node b_i, with $0 \leq i \leq n$ and $a < b \leq n$, when the node visited just before b_i is a_k. Note that the arc (a_k, b_i) is a block1 arc in G, which is defined only when $A_a + T < A_b$ and $P_a \neq P_b$.

In the graph constructed in Section 2.3, there are $O(n^2)$ arcs leaving nodes from layer $L(a)$: $O(n)$ block2 arcs and $O(n^2)$ block1 arcs. Clearly, a shortest path to some node b_l has as last arc either a block1 arc, or some arc that is not a block1 arc. This is reflected in the following expression for the length of a shortest path from σ to some node $b_l \in L(b)$.

$$sp(b_l) = \min \left(\min_{\substack{a<b \\ k=0,\ldots,n}} sp^{a_k}(b_l),\ \min_{a<b} sp^{a_{top}}(b_l),\ sp^{\sigma}(b_l) \right).$$

In the following, we argue that once $sp(a_k)$ is determined for an arbitrary a and for each $k \in \{= 0, \ldots, n\}$, all $sp^{a_k}(b_l)$, i.e. shortest paths determined by block1 arcs leaving $L(a)$, can be obtained in $O(n)$ time.

For a given b_l, it holds that $sp^{a_k}(b_l) = sp(a_k) + c(a_k, b_l)$, whereby $c(a_k, b_l) = w(a, b) + k(A_b - A_a - T)$ is the length of the block1 arc (a_k, b_l) in G. The term $w(a, b)$ represents the waiting time accumulated at time $A_b + T$ of all ships arriving in $(A_a, A_b + T]$ at position $1 - P_b$ and the waiting time of all ships arriving in $(A_a + T, A_b]$ at position P_b; this is a constant over all a_k, b_l for which a block1 arc (a_k, b_l) exists. Note that for

each a_k, there is at most one b_l from layer $L(b)$ for which an arc (a_k, b_l) exists.

We argue that, for any given a, finding all $sp^{a_k}(b_l)$ where $0 \leq k \leq n$ and $a < b \leq n$, corresponds to finding the minimum value of at most n linear functions ($\forall k = 1, \ldots, n$) for no more than n inputs ($\forall b = a+1, \ldots, n$). Note that l is fixed, given a_k and b.

Lemma 2.3. *Given m linear functions $f_i(x) = \alpha_i \cdot x + \beta_i$ with $\alpha_i, \beta_i \in \mathbb{R}$ for $i \in \{1, \ldots, m\}$, with $\alpha_i \leq \alpha_{i+1}$ for each $i = 1, \ldots, m-1$, and given an ordered finite set Q. Then the indices $\arg\min\{f_i(q) \mid i = 1, \ldots, m\}$ for each $q \in Q$ can be found in $O(m + |Q|)$.*

Proof. It is clear that we may assume, without loss of generality, that no two of the given functions are identical. We start by eliminating dominated functions. We say that f_i is dominated by f_j, $j < i$, if $\beta_j \leq \beta_i$ because then $f_j(q) \leq f_i(q)$ for each $q \geq 0$. Henceforth, we assume that $\beta_i > \beta_{i+1}$ for each $i = 1, \ldots, m-1$ in the following.

Next, we find the lower envelope of the functions $f_1, \ldots, f_m$. We refer to Sack and Urrutia (2000) for an overview of methods devoted to computing the lower envelope of a set of functions. We maintain a list $\mathcal{E}$ of active functions which potentially contribute to the lower envelope. Note that functions may be dropped from this list later on. Initially, this list contains f_m and f_{m-1}. We consider the functions f_{m-2}, f_{m-3}, $\ldots$, f_1 in this order. Let f_i be the function to be considered, and f_w and f_v the last and next-to-last function contained in $\mathcal{E}$, respectively. We determine the q-coordinate $q_{i,w} \equiv \frac{\beta_i - \beta_w}{\alpha_w - \alpha_i}$ of the intersection of f_i and f_w and the q-coordinate $q_{i,v} \equiv \frac{\beta_i - \beta_v}{\alpha_v - \alpha_i}$ of the intersection of f_i and f_v. If $q_{i,v} \geq q_{i,w}$, then we remove f_w from $\mathcal{E}$ since $f_w(q) \geq \min(f_i(q), f_v(q))$ for each $q \geq 0$, and we repeat this step with $\mathcal{E} := \mathcal{E} \setminus \{f_w\}$ until $q_{i,v} < q_{i,w}$. We then add f_i to $\mathcal{E}$, i.e. let $\mathcal{E} := \mathcal{E} \cup \{f_i\}$, and consider the next function. Note that at the end of each iteration, $\mathcal{E}$ must contain at least two functions since f_m is not dominated due to assumption.

Finally, by scanning through the sorted set of intersections of linear functions and through values in Q, we find $\arg\min\{f_i(x) \mid i = 1, \ldots, m\}$ for each $q \in Q$. For a graphical representation of the lower envelope, see Figure 2.8.

It remains to show that this procedure runs in linear time. Eliminating dominated functions takes $O(m)$. During the construction of $\mathcal{E}$, each function is added at most once and each function is deleted at most once. The decision whether to add the next function at the end of $\mathcal{E}$ or remove

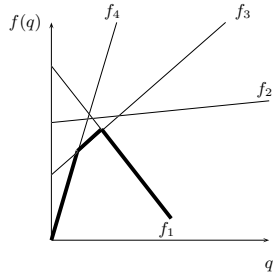


Figure 2.8: Lower envelope

the currently last function from $\mathcal{E}$ can be taken in constant time. Thus, constructing $\mathcal{E}$ takes $O(m)$ and, consequently, finding $\arg\min\{f_i(q) \mid i = 1, \ldots, m\}$ for each $q \in Q$ takes $O(m + |Q|)$. □

We now use Lemma 2.3 and consider all block1 arcs leaving $L(a)$, for a given $a \in \mathcal{S}$. Consider the linear functions $sp^{a_k}(b_l) = sp(a_k) + w(a, b) + k(A_b - A_a - T)$, for $k \in \{0, \ldots, n\}$ and for all b with $a < b \leq n$. As mentioned before, $w(a, b)$ is a constant for any given a and b. We thus exclude it from the linear functions to be considered for the lemma, and add this constant later. Let $Q = \{A_b - A_a - T \mid b = a + 1, \ldots, n\}$ and $f_k(q) = sp(a_k) + q \cdot k$ for each $0 \leq k \leq n$. Thus, referring to Lemma 2.3, $\alpha_k = k$, $\beta_k = sp(a_k)$, and $q \in Q$. In Lemma 2.3 it is assumed that the α_i and the set Q are ordered, which is obviously the case. Now, $\arg\min\{sp^{a_k}(b_i) \mid 0 \leq k \leq n\}$ for each b_i determines the shortest path to b_i having a block (a_k, b_i) of length one as last part. Note that its length is $\min\{f_k(q) \mid 0 \leq k \leq n\} + w(a, b)$. Hence, we can update all $sp(b_i)$ taking into account all block1 arcs leaving a layer $L(a)$ in $O(n)$ time.

Applying this procedure to all layers $L(a)$, with $a \in \mathcal{S}$, all shortest paths with correct cost values are obtained in $O(n^2)$. The overall procedure to find the shortest path in $O(n^2)$ is shown in Algorithm 1. We can summarize the above in the following theorem.

Theorem 2.2. *The lockmaster's problem is solvable in $O(n^2)$.*

Create the set of nodes V as described above
$sp(s_k) \leftarrow \infty$ for all $s \in \mathcal{S}$ and $0 \leq k \leq n$
for $s = 1, \ldots, n$ **do**
  Consider block (σ, s), update $sp(s_k)$ for k corresponding to (σ, s)
  Set $sp(s_{\text{top}}) = \min \{sp(s_k) \mid 0 \leq k \leq n\}$
  Determine the lower envelope for all block1 arcs (s_k, s'_l)
  for $s' = s + 1, \ldots, n$ **do**
    Determine the l for which $(s_k, s'_l) \in E$
    if $\ell(s, s') == 1$ **then**
      Update $sp^{s_k}(s'_l)$ using the lower envelope
    else
      Update $sp^{s_k}(s'_l)$ using $sp(s_{\text{top}})$

Algorithm 1: $O(n^2)$ algorithm for finding the shortest path in G

2.5 Extensions

It can be argued that the basic problem defined in Chapter 1, due to different assumptions regarding the input, ignores a number of issues regarding practical lock operation. We now proceed by showing how the procedure described in Section 2.3 can be extended towards a closer approximation of reality.

2.5.1 Capacity

Section 2.3 did not take any capacity restrictions into account. We now discuss two different settings with capacity restrictions. In one setting, the lock can accommodate at most a given number of ships; in another setting each ship has a size, the total size of ships within a lock should then not exceed the given lock's size. A fundamental difference with the basic lockmaster's problem is that ships may be "left behind". Thus, ships that lay waiting may not all be transferred in the same lockage.

When the number of ships that may be contained within the lock at any given time is bounded by a constant, a similar procedure to Section 2.3 may be used.

Theorem 2.3. *The lockmaster's problem with an upper bound on the number of ships in the lock is solvable in* $O(n^4)$.

Proof. Suppose that the lock can accommodate at most C ships at once. We can reformulate Property 2.3 as follows: the final lockage does not end later than $(1 + \lceil \frac{n}{C} \rceil)2T$. For this generalization, Lemma 2.1 remains true. Indeed, Properties 2.1 and 2.2 remain trivially true; moreover, the ships being identical (except for their arrival time) implies that there exists an optimal schedule that satisfies Property 2.4.

Definition 2.3.

- *For each* $a \in \mathcal{S}$, *let* $\mathcal{U}_a = \{b \in \mathcal{S} \mid P_b = 1, b < a, A_b \leq A_a - |P_b - P_a|T\}$.
- *For each* $a \in \mathcal{S}$, *let* $\mathcal{D}_a = \{b \in \mathcal{S} \mid P_b = 0, b < a, A_b \leq A_a - |P_b - P_a|T\}$.

Thus, the set $\mathcal{U}_a$ (respectively $\mathcal{D}_a$) contains all ships b for which $b < a$, that arrive on the upstream (downstream) side not later than A_a if ship a arrives on the upstream (downstream) side, and not later than $A_a - T$ if ship a arrives on the downstream (upstream) side.

We now design the following directed graph $G = (V, E)$ in order to represent the problem. In contrast to the graph used for the lockmaster's problem, here, each arc directly corresponds to a set of ships transferred, and there is no need to distinguish block1 and block2 arcs. We have $V = \{\sigma, \tau\} \cup V^A$ with $V^A = \{(a, u, d) \mid a \in \mathcal{S}, u \in \mathcal{U}_a, d \in \mathcal{D}_a\}$. A node (a, u, d) corresponds to a block starting at A_a and position P_a while u and d represent the latest served (with respect to the ordering in $\mathcal{S}$) downstream- and upstream-bound ship, respectively. It is easily seen that whenever a ship a does not enter a lockage starting not earlier than A_a from position P_a while sufficient lock capacity is available to serve this ship, the solution is suboptimal. In combination with Property 2.4 it follows that, within each block, each ship is served by the next appropriate lockage with sufficient capacity, and it is thus known when each ship is handled. We thus know which ships u' and d' become the latest ships served upstream and downstream, respectively. We record this observation by writing $u' = f_{\text{up}}((a, b), u)$ and $d' = f_{\text{down}}((a, b), d)$ for each block (a, b), $u \in \mathcal{U}_a$, $d \in \mathcal{D}_a$.

Definition 2.4. *We say that two nodes* (a, u, d) *and* (b, u', d') *are* compatible *if* (a, b) *is a block and if* $u' = f_{up}((a, b), u)$ *and* $d' = f_{down}((a, b), d)$.

The set of edges in our graph is given as $E = E^\sigma \cup E^B \cup E^\tau$, with

- $E^\sigma = \big\{(\sigma, v) \mid v = (a, \emptyset, \emptyset) \in V^A\big\}$,

- $E^B = \{(v, v') \mid v \text{ and } v' \text{ are compatible}\}$, and
- $E^\tau = \{(v, \tau) \mid v = (a, u, d) \in V^A\}$.

The cost $c(v, v')$ of the edges in E^B, with $v = (a, u, d)$ and $v' = (b, u', d')$, equals the sum of waiting times of all ships k with $u < k \leq u'$ arriving at position $p = 1$ and the sum of waiting times of all ships l with $d < l \leq d'$ arriving at position $p = 0$, such that these ships are handled while respecting the lock capacity.

The costs $c(s, v)$ on edges leaving the origin are equal to 0, since no ship has been transferred yet. The costs $c(v, t)$, with $v = (a, u, d)$, represent the waiting times of ships k with $u < k$ arriving at position $p = 0$ and the waiting times of ships l with $d < l$ arriving at position $p = 1$. These ships are served by the final sequence of lockages.

Then, a path from σ to τ represents a feasible lock schedule and the shortest path represents the lock schedule having minimum total waiting time of ships. G is acyclic and contains $O(n^3)$ nodes, each node being source of at most $O(n)$ arcs. Furthermore, we can determine u', d', and total waiting of ships according to all arcs emerging from $v \in \{s\} \cup V^A$ in $O(n)$ time. The total time complexity for this algorithm is thus $O(n^4)$. □

A more general setting assigns to each ship and lock an arbitrary size. When each ship and lock has an associated length and width, the problem is easily seen to be NP-hard by reduction from rectangle packing, as mentioned by Hermans (2014). We extend this result to instances where the size of the lock is represented by a scalar value. Thus, at any given time, the sum of size values of the ships in the lock must not exceed the size of the lock.

Theorem 2.4. *The lockmaster's problem with a (scalar) bound on the size of the lock is strongly NP-hard, even in the uni-directional setting and when all ships arrive at time $t = 0$.*

Proof. We provide a proof by reduction from 3-PARTITION. In an instance of 3-PARTITION, we are given an integer B and a set A consisting of integers a_i $(i = 1, \ldots, 3n)$ subject to $\frac{B}{4} < a_i < \frac{B}{2}$ for each i. The question is whether A can be partitioned into n triples such that the sum of integers in each triple equals B. This problem is known to be NP-complete. We now construct an instance of the lockmaster's problem where the size of a ship is encoded by the number of the 3-PARTITION instance, all ships arrive at $t = 0$ in position $p = 1$, and the lock's size encodes the value B to be met by each triple. Clearly, if the instance of 3-PARTITION has

a solution, then there exists a lock schedule where n lockages each serve exactly 3 ships. Each of these lockages is followed by an empty lockage that returns the lock to position $p = 1$, resulting in a solution with a total waiting time of $3n(n-1)T$. On the other hand, any solution with a total waiting time of $3n(n-1)T$ such that at most 3 ships can be handled in each subsequent $2T$ interval, will transfer all ships in the first n lockages starting from position $p = 1$. Consequently, there must exist n subsets of ships so that each subset fills the lock completely, and thus a solution exists to the instance of 3-PARTITION. □

We may, however, impose the requirement that all ships travelling in the same direction must be handled in a predetermined order. More specifically, we will require that ships travelling in the same direction are handled according to the total order on $\mathcal{S}$. We will say that solutions satisfying this requirement adhere to a first-come first-served (FCFS) policy. The FCFS policy is currently applied as the standard handling policy for ships at many locks, see e.g. Smith et al. (2009); Ting and Schonfeld (2001); van Haastert (2003). When this requirement is enforced, an efficient procedure exists for finding an optimal solution.

Theorem 2.5. *Under a FCFS policy, the lockmaster's problem with a bound on the size of the lock is solvable in $O(n^4)$.*

Proof. In a preprocessing step, the subsets of consecutive ships that fit together in the lock, are determined. Here, it is possible to include all kinds of filling and entering rules that can be important in practice, see e.g. Verstichel and Vanden Berghe (2009). This step can be executed in $O(n^2)$ time. We use the same graph as defined for the setting with bounded capacity, described above. Note that when determining u', d', and total waiting time of ships according to arcs, we can only fill the lock with ships that fit together in the lock, which we determined initially. Still, we can treat all arcs emerging from a node in $O(n)$ time. □

Notice that although in the instance constructed in the proof of Theorem 2.4 all ships arrive at the same time, it is not true that every sequence of ships respects FCFS; only the sequence that is compatible with the given order on $\mathcal{S}$ adheres to the FCFS policy. This explains why Theorems 2.4 and 2.5 are not contradictory.

2.5.2 Ship priorities

In this subsection we are interested in minimizing the weighted sum of waiting times; this allows us to take into account ship priorities. This is often relevant, for instance, in locks operating in ports. It is indeed quite common to distinguish between sea ships (having limited manoeuvrability) and inland ships. In addition, ships transporting dangerous goods receive priority over regular cargo ships (Du and Yu, 2003), which in turn may have priority over leisure ships, see e.g. (Smith et al., 2009; Verstichel and Vanden Berghe, 2009). All this can be dealt with by assigning a weight w_a to each ship $a \in \mathcal{S}$, revealing their priority.

It is not hard to see that we can generalize the construction of the graph G, and in particular the arc-costs, to this setting by multiplying the waiting time of ship a by its weight w_a. We state without proof:

Theorem 2.6. *The lockmaster's problem with objective to minimize total weighted waiting time is solvable in* $O(n^3)$.

Notice that when the ships are weighted and the capacity of the lock is limited, the algorithm described in Section 2.5.1 might fail to find an optimal solution. Earlier, specifying a block would implicitly specify the set of ships that are transferred in this block. This, however, need then no longer be the case; although the optimal solution can still be seen as consisting of blocks, it is not clear which particular ships are transferred during which lockages in a block. When considering the uni-directional case, Baptiste's algorithm (Baptiste, 2000) (see Section 2.2) yields a polynomial time procedure.

2.5.3 Handling times

In practice, placing a ship into a lock takes a certain amount of time. The total time a ship spends in the lock may then depend on the other ships present. The required handling time per ship may be constant, which is approximately the case for inland ships. For sea ships, on the other hand, the handling time depends on the size and manoeuvrability of the ship. In Smith et al. (2011), a lock scheduling problem is described where handling times are not only ship-dependent, but also sequence dependent. In this section, we consider the extension where each ship $s \in \mathcal{S}$ requires a certain, integral, handling time h_s. We model this by modifying the lockage time so that it is no longer constant, but depends on the ships that are present in the lockage: $T_j = T + \sum_{s \in \mathcal{S}_j} h_s$, where $\mathcal{S}_j$ refers to the set of ships

transferred in the j-th lockage, and T_j denotes the corresponding lockage duration, $j = 1, 2, \ldots$.

Observe that, due to possibly distinct h_s values, it may be optimal to let a ship with a relatively large handling time wait while handling subsequent arrivals first. Thus optimal solutions may involve the overtaking of ships, and hence violate Property 2.4. We show that the lockmaster's problem with ship dependent handling times is NP-hard. However, as mentioned earlier, the FCFS policy is a reasonable additional requirement when dealing with lock scheduling problems. Also notice that, even when solutions must satisfy FCFS, it might be beneficial to let a ship with a relatively large handling time wait, so that the lock is able to move quickly to the other side and transfer ships that have arrived there. Nevertheless, we show that for this case a polynomial time algorithm exists. Note that this also applies to the setting where all handling times are equal, i.e. the case where $h = h_s$ for all $s \in \mathcal{S}$.

It may also be worth noting that by introducing these ship-dependent handling times, an alternative definition for waiting time arises. Recall that the waiting time for a ship $s \in \mathcal{S}$ was defined in Chapter 1 as the difference between the starting time of the lockage containing s, and the arrival time of s. However, when handling times are present, we may define the waiting time for a ship to be the difference between the actual and the earliest possible time where that ship could leave the lock, i.e. for ship $s \in \mathcal{S}$ this waiting time corresponds to the ending time of the lockage containing ship s, reduced by the arrival time A_s, the fixed part T of the lockage time, and the handling time h_s. Note that, with this definition of waiting time, even if ship s enters the lock at time A_s, it may still incur a positive waiting time due to the presence of other ships in the same lockage. When all handling times are equal to zero, both definitions result in the same value for each ship. In the remainder of this section, we will continue to use our original definition of waiting time.

We first show that the general setting for the lockmaster's problem with handling times is strongly NP-hard.

Theorem 2.7. *The lockmaster's problem with ship-dependent handling times is strongly NP-hard, even in the uni-directional setting.*

Proof. We provide a reduction from 3-PARTITION (see the proof of Theorem 2.4 for the definition). For any given instance of 3-PARTITION, we construct a corresponding instance of the lockmaster's problem with ship-dependent handling times. We argue that solving the latter instance to optimality allows to decide the question of the 3-PARTITION instance.

The lockmaster's instance is constructed as follows: Let $T = 1$, although it can be seen that an instance can be constructed for arbitrarily chosen T. At time $t = 0$, a set of $3n$ ships arrives on the downstream side, with for each ship a handling time $h_i = a_i$ $(i = 1, \ldots, 3n)$. We refer to these ships as the *handling ships*. At each of the times $t = B + 2T, t = 2B + 4T$, $\ldots$, $t = n(B + 2T)$, a set of $3(B + 2T)n^2$ ships arrives on the downstream side. Each of these ships has zero handling time, i.e. $h_j = 0$ for $j = 3n + 1, \ldots, 3n + 3(B + 2T)n^3$. We refer to these ships as the *blocking ships*. The question is the following: "does there exist a solution with total waiting time not greater than $W \equiv 3(B+2T)n(n-1)/2$?" This completes the description of the instance of the lockmaster's problem with handling times. Figure 2.9 provides a graphical illustration of this instance.

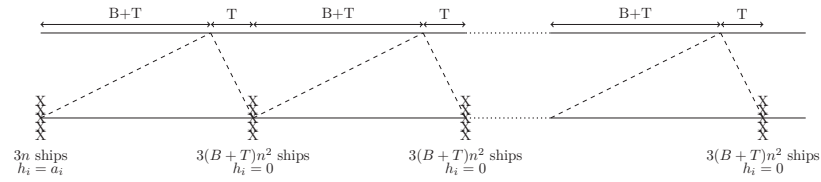


Figure 2.9: Illustration of the instance for the proof of Theorem 2.7. The dashed lines correspond to the lockages in a feasible solution for a 'yes' instance.

We now argue that the existence of a solution to the lockmaster's instance with total waiting time not greater than W implies a 'yes' answer to the 3-PARTITION question. Observe that since all data are integral, we can assume that the lock moves only at integral moments in time. Then from $W < 3(B + 2T)n^2$ it follows that any solution with value no greater than W must serve all blocking ships at their arrival time. Hence, for all feasible solutions with a total waiting time not greater than W, the lock moves from the downstream side to the upstream side at times $t = B + 2T$, $t = 2B + 4T$, $\ldots$, $t = n(B + 2T)$. The lock thus has n intervals of size $B + 2T$ to handle the ships that arrive at $t = 0$. Because, for each lockage, a duration of T cannot be avoided regardless of the number of ships in the lock, it follows that the sum of handling times of all ships in each upward lockage can be at most equal to B. Since $\frac{B}{4} < a_i$, it is impossible to handle four ships in such an interval. As a result, the only way to achieve

a total waiting time not greater than $\sum_{i=1}^{n} 3(n-i)(B+2T)$, is to handle three handling ships in each of the n intervals with length $B+2T$. Note that $\sum_{i=1}^{n} 3(n-i)(B+2T) = 3(B+2T)(n-1)n/2 = W$. This, however, implies that the sum of handling times in each upwards lockage is exactly equal to B, and thus a corresponding solution to the initial 3-PARTITION instance exists. Thus, we can verify whether an instance of 3-PARTITION is a 'yes'-instance by solving the corresponding lockmaster's instance with ship-dependent handling times and checking whether the total waiting time is no more than W. Conversely, it is obvious that if a partition exists where all triples sum up to B, there is also a lock schedule with total waiting time equal to W. Thus, if the optimal solution value to the lockmaster's instance is strictly greater than W, this identifies the given instance as a 'no'-instance. □

Note that when the waiting time for a ship is defined according to the alternative definition above, where the increase in lockage duration due to the presence of other ships is considered as waiting time, we can easily modify the proof so that the theorem remains valid. Indeed, we may choose $T > \sum_{i=1}^{3n} h_i$, increase the number of blocking ships arriving at $t = B+2T$, $t = 2B+4T$, $\ldots$, $t = n(B+2T)$ by $2\sum_{i=1}^{3n} h_i$, and let $W \equiv 3(B+2T)n(n-1)/2 + 2\left(\sum_{i=1}^{3n} h_i\right)$. Observe that because 3-PARTITION is NP-complete in the strong sense, we may assume that the values a_i are bounded by a polynomial in n; the size of the corresponding lockmaster's instance thus also remains polynomial in n. Because the value for T was chosen to be sufficiently large, any solution where an upwards lockage contains two or less ships necessarily has an objective value larger than W. Each lockage starting in position $p = 0$ in a solution with value not greater than W must thus contain exactly 3 ships. It then follows that, in all such solutions, a waiting time of $2\left(\sum_{i=1}^{3n} h_i\right)$ is incurred due to ships being simultaneously present within the lock. The remaining argumentation for the proof remains unchanged since the redefinition of W reflects this increase in waiting time.

Similar to the setting with capacity and arbitrary ship sizes (Section 2.5.1), which is also NP-hard in general, we observe that a polynomial time procedure exists when restricting the solutions to those which adhere to the FCFS policy.

Theorem 2.8. *The lockmaster's problem with arbitrary handling times under a FCFS policy can be solved in $O(n^{10})$ time.*

Proof. Observe that the arguments used to establish Lemma 2.1 in Section 2.3 also apply to the case of arbitrary handling times. Hence Properties 2.1 and 2.2 remain valid in this setting.

We now construct a graph $G = (V \cup \{\sigma, \tau\}, E)$ as follows. For each ship $s \in \mathcal{S}$, and for each possible number of ships waiting upstream at time A_s (referred to as w_u), and for each possible number of ships waiting downstream at time A_s (referred to as w_d), we create a node (also called state) $v = (s, w_u, w_d)$ in V. Notice that s is not included in either w_u or w_d. To define the edges of G, we consider each pair of states (v, v') where $v' = (s', w'_u, w'_d)$, with $A_{s'} > A_s + T$ if $P_s \neq P_{s'}$ or with $A_{s'} > A_s + 2T$ if $P_s = P_{s'}$. It is important to realize that, when given two states v and v', it is exactly known which ships are transferred when moving from state v to state v'. Indeed, the FCFS assumption, together with the number of ships waiting at A_s and $A_{s'}$ directly reveals which ships have been transferred. We will denote this set of ships by $\mathcal{S}(v, v') \subseteq \mathcal{S}$.

We define a *handling strategy* from v to v' as a set of consecutive lockages that start at A_a, ends in the interval $(A_{a'} - 2T, A_{a'}]$, and collectively transfer the set of ships in $\mathcal{S}(v, v')$. Notice that a handling strategy need not exist; in addition, there may be different handling strategies for a given pair of states (v, v'). However, the following is true for any handling strategy (if one exists):

Lemma 2.4. *Given a pair of states $v = (s, w_u, w_d)$ and $v' = (s', w'_u, w'_d)$, any feasible handling strategy consists of the same number of lockages.*

Proof. Since we know v and v', we know which ships are transferred by any handling strategy, and hence we know the total handling time: $H(v, v') = \sum_{s \in \mathcal{S}(v,v')} h_s$. By Property 2.2 above, we also know that the ending time of the last lockage of the strategy cannot exceed $A_{s'}$, nor can it end earlier than $tA_{s'} - 2T$. Thus, in case $P_s = P_{s'}$, we look for an even number of lockages, called $m(v, v')$, that satisfies:

$$A_{s'} - 2T < A_s + H(v, v') + m(v, v')T \leq A_{s'}.$$

Similarly, in case $P_s \neq P_{s'}$, we look for an odd number of lockages $m(v, v')$ satisfying the equation above. Due to Property 2.2, the value of $m(v, v')$ is unique and the lemma follows. □

For all pairs v, v' for which a number $m(v, v')$ can be computed, we have an edge in G. Further, there is an edge from node σ to each state v, and there is an edge from each state v to node τ.

We now turn to describing how the costs of the edges in E are chosen. To compute the costs of an edge from v to v', we will invoke a procedure that computes the cost of the corresponding optimal handling strategy from v to v'. We point out that it may still be the case that, although a number $m(v, v')$ can be computed, no feasible handling strategy exists. This will follow from the procedure that we describe next. The resulting cost of that edge is then set to ∞. A handling strategy from v to v' will specify for each ship in $\mathcal{S}(v, v')$ in which lockage of the handling strategy it will be transferred.

The cost of a handling strategy consists of the waiting time of relevant ships. A ship a with $A_a \leq A_{s'}$ is relevant when it has not yet been transferred at time A_s. For each relevant ship a, let t_a be the time at which a is served by the lock. The waiting time for a, within this handling strategy, then equals the time that elapses between $\max(A_s, A_a)$ and $\min(t_a, A_{s'})$.

We now show how to compute the minimum total waiting time of the optimal handling strategy from v to v'. This will become the cost of the arc from state v to v' in the graph G. Recall that $\mathcal{S}(v, v')$ is the set of ships transferred; we write $\mathcal{S}(v, v') = \mathcal{S}^u(v, v') \cup \mathcal{S}^d(v, v')$, where $\mathcal{S}^u(v, v')$ and $\mathcal{S}^d(v, v')$ respectively refer to those ships in $\mathcal{S}(v, v')$ that arrive on the upstream and downstream side. Let $u(k)$ and $d(k)$ represent the k'th element in $\mathcal{S}^u(v, v')$ and $\mathcal{S}^d(v, v')$ respectively.

Given v and v', we construct a graph H. There is a node in H for each state (k_u, k_d, m); k_u ranges from $0, \ldots, |\mathcal{S}^u(v, v')|$, k_d ranges from $0, \ldots, |\mathcal{S}^d(v, v')|$, and $m = 1, \ldots, m(v, v')$. Note that $m(v, v')$ can be uniquely determined as a result of Lemma 2.4. Thus, a state (k_u, k_d, m) represents the situation that the first k_u upstream ships from $\mathcal{S}(v, v')$, as well as the first k_d downstream ships from $\mathcal{S}(v, v')$ have been transferred using m lockages. In addition, we add a starting node $(0, 0, 0)$. Let us now consider the edges in H. For any vertex $w = (k_u, k_d, m)$, we find all possible states $w' = (k'_u, k'_d, m+1)$ that can be reached by the next lockage. Notice that this includes an empty lockage, i.e. state $(k_u, k_d, m + 1)$. Also, a single lockage cannot serve both upstream and downstream ships: thus, only states with either $k'_u = k_u$ or $k'_d = k_d$ can be reached. More formally, we associate to each state a so-called ending time $T(k_u, k_d, m)$, which can be computed as follows:

$$T(k_u, k_d, m) = t(s) + mT + \sum_{a \in \mathcal{S}_{k_u, k_d}(v, v')} h_a,$$

where $\mathcal{S}_{k_u,k_d}(v,v')$ corresponds to the first k_u upstream ships, and the first k_d downstream ships from $\mathcal{S}(v,v')$. Observe that $T(0,0,0) = A_s$.

We now specify the arcs in H. Observe that since both P_a and m are known, we can infer whether an arc from node (k_u, k_d, m) to node $(k'_u, k'_d, m+1)$ represents an upward or a downward lockage of the lock. Without loss of generality, let us consider a downward lockage of the lock: we draw an arc from node (k_u, k_d, m) to node $(k'_u, k_d, m+1)$ for all k'_u that satisfy $t_u(k'_u) \leq A_q$, where q is the latest ship to arrive upstream before $T(k_u, k_d, m)$. A similar construction holds for any upward lockage of the lock. Since we know which ships we transfer for each arc in H, we can compute the total waiting time of the relevant ships in this lockage. This number is the cost of this edge, and we have specified the graph H.

We claim that a shortest path in H from $(0,0,0)$ to $(|\mathcal{S}^u(v,v')|, |\mathcal{S}^d(v,v')|, m(v,v'))$, if it exists, corresponds to a minimum-cost handling strategy from v to v'. Indeed, this claim follows from the observation that any way of transferring the ships in $\mathcal{S}(v,v')$ is a path in H with the corresponding waiting time, and vice versa. Furthermore, we argue that such a shortest path can be found in $O(n^4)$ time. We introduce the following lemma which reduces the number of handling strategies that we need to consider:

Lemma 2.5. *All handling strategies where two consecutive empty lockages are followed by a nonempty lockage can be excluded without impacting the optimality of the resulting solution.*

Proof. Assume, without loss of generality, that the non-empty lockage starts on the downstream side. Notice that the handling strategy is non-optimal if any ship moved by the non-empty lockage arrives before the first empty lockage. We thus assume the existence of at least one downstream arrival during the empty lockages. It is then easy to see that there is an optimal block schedule with a block starting at that arrival, and thus excluding the two empty lockages. □

Note that for all m larger than $2|\mathcal{S}(v,v')| + 1$, either two consecutive empty lockages are followed by a non-empty lockage, or all ships in $\mathcal{S}(v,v')$ have been moved. The values for m that we need to consider are thus bounded by $O(n)$ so that we find a shortest path in H considering only $O(n^3)$ nodes and $O(n^4)$ arcs.

Correctness of the theorem thus follows from the fact that an optimal solution exists that displays the structure of Lemma 2.1 and since we compute, for each possible pair of states, a handling strategy with minimum

cost. Since any σ-τ path in G represents a feasible lock schedule, the shortest path in this graph corresponds to the optimal lock schedule. With respect to the complexity of the procedure we observe by Lemma 2.5 that the number of states to be considered in G equals $O(n^3)$, and hence the number of edges equals $O(n^6)$. Since a shortest path in H can be found in $O(n^4)$ steps, we have a polynomial time bound of $O(n^{10})$. □

2.5.4 Non-uniform lockage duration

It is not uncommon that the lockage duration for an upwards lockages differ from the lockage duration for a downwards lockage, for example due to the current. We argue that the procedure for the lockmaster's problem outlined in Section 2.3 is easily adjusted to take this into account.

Theorem 2.9. *The lockmaster's problem with a lockage duration that depends on the position of the lock is solvable in $O(n^3)$.*

Proof. We redefine a block $B(a, b)$ from Section 2.3 as follows.

Definition 2.5.

- *For each $a, b \in \mathcal{S}$ with $P_a = 0 \neq P_b$ and with $A_a + T_u < A_b$, a block $B(a, b)$ is the set of $\ell(B)$ consecutive lockages that starts at A_a and ends at $A_a + \lceil \frac{\ell(B)}{2} \rceil T_u + \lfloor \frac{\ell(B)}{2} \rfloor T_d$; with $\ell(B)$ the largest odd integer such that $A_a + \lceil \frac{\ell(B)}{2} \rceil T_u + \lfloor \frac{\ell(B)}{2} \rfloor T_d \leq A_b$.*

- *For each $a, b \in \mathcal{S}$ with $P_a = 1 \neq P_b$ and with $A_a + T_d < A_b$, a block $B(a, b)$ is the set of $\ell(B)$ consecutive lockages that starts at A_a and ends at $A_a + \lceil \frac{\ell(B)}{2} \rceil T_d + \lfloor \frac{\ell(B)}{2} \rfloor T_u$; with $\ell(B)$ the largest odd integer such that $A_a + \lceil \frac{\ell(B)}{2} \rceil T_d + \lfloor \frac{\ell(B)}{2} \rfloor T_u \leq A_b$.*

- *For each $a, b \in \mathcal{S}$ with $P_a = P_b$ and with $A_a + T_u + T_d < A_b$, a block $B(a, b)$ is the set of $\ell(B)$ consecutive lockages that starts at A_a and ends at $A_a + \frac{\ell(B)}{2}(T_u + T_d)$; with $\ell(B)$ the largest even integer such that $A_a + \frac{\ell(B)}{2}(T_u + T_d) \leq A_b$.*

It is not difficult to verify that all results from Section 2.3, mutatis mutandis, apply to this setting. □

2.5.5 Water usage

Due to organizational or environmental reasons, there can be a limit on the number of lockages allowed in a given time-interval. In particular, when water is scarce (e.g. after dry seasons), lockages may seriously disrupt the water level. In such a case, the number of allowed lockages is bounded in order to keep the water at a navigable level (Verstichel and Vanden Berghe, 2009). Again a modification to the original procedure can be found which finds an optimal solution. This procedure runs in polynomial time provided that the bound Q on the number of lockages is part of the input.

Theorem 2.10. *The lockmaster's problem with an upper bound Q on the total number of lockages is solvable in $O(Q^2n^4)$.*

Proof. Let Q refer to the maximum number of allowed lockages. Notice that in this situation Lemma 2.1 no longer holds. Indeed, a lockage will no longer occur at least every $2T$ interval. However, it does hold that lockages are either consecutive or start upon arrival time of a ship; and the final lockage ends no later than $A_n + 3T$. We now define a *bounded block* $B = (a, b, \ell(B))$ for each $a, b \in \mathcal{S}, \ell(B) \leq Q$ with $A_a + \ell(B)T < A_b$, as a set of $\ell(B)$ consecutive lockages starting at A_a, with $\ell(B)$ even if $P_a = P_b$, and $\ell(B)$ odd if $P_a \neq P_b$. It must hold that $A_a + \ell(B)T < A_b$.

Again, $\ell(B)$ is called the length of block B. We do not prove it formally but it should be clear that each schedule can be represented by a sequence of bounded blocks $B_1, \ldots, B_\alpha$ where $B_k = (a_k, b_k)$, $k = 1, \ldots, \alpha$ with $a_k, b_k \in \mathcal{S}$ and $b_k = a_{k+1}$ for each $k = 1, \ldots, \alpha - 1$.

If we define U_a and D_a as in Section 2.5.1, we may design the following directed graph $G = (V, E)$ in order to represent the problem. We have $V = \{s, t\} \cup V^A$ with $V^A = \{(a, u, d, q) \mid a \in \mathcal{S}, u \in U_a, d \in D_a, 0 \leq q \leq Q\}$. A node (a, u, d, q) corresponds to a bounded block starting at A_a, while u and d represent the latest ships served coming from upstream and downstream, respectively, and q denotes the number of finished lockages. It holds that:

$$u' = f_{\text{up}}((a, b, l(a, b)), u); d' = f_{\text{down}}((a, b, l(a, b)), d)$$

Definition 2.6. *We say that two nodes (a, u, d, q) and (b, u', d', q') are* compatible *if (a, b) is a block and if $u' = f_{up}((a, b, l(a, b)), u)$ and $d' = f_{down}((a, b, l(a, b)), d)$, and $q' = q + l(a, b)$.*

The set of edges is given as $E = E^s \cup E^B \cup E^t$, with

- $E^s = \big\{(s, v) \mid v = (a, \emptyset, \emptyset, 0) \in V^A\big\}$,
- $E^B = \{(v, v') \mid v \text{ and } v' \text{ are compatible}\}$, and
- $E^t = \big\{(v, t) \mid v = (a, u, d, q) \in V^A \text{ with } q + \kappa \leq Q$ where κ is the shortest length of a bounded block starting at A_a and serving all ships not transported yet $\big\}$.

The meanings of nodes and arcs are analogous to the ones in Section 2.5.1. The only difference is that the number of lockages is encoded in nodes here and arcs represent bounded blocks instead of blocks. Graph G now contains $O(n^3Q)$ nodes. Note that each node is source of $O(Qn)$ arcs. We can determine $u'((a, b, l(a, b)), u)$ and $d'((a, b, l(a, b)), d)$, and total waiting time of ships according to all arcs emerging from $v \in \{s\} \cup V^A$ in $O(Qn)$ time.

Obviously, a path from σ to τ then represents a feasible lock schedule and the shortest path represents the lock schedule having minimum total waiting time of ships. Since G is acyclic the theorem follows. □

2.6 Heuristics

Let us also define a number of heuristics that solve the single-lock single-chamber scheduling problem. In Section 2.7, we then compare the performance of the exact algorithms to the heuristics presented below.

Continuous up/down (CUD): The lock will move whenever possible, resulting in a continuous up/down movement. At the start of each lockage, as many ships are moved as the lock capacity (if limited) allows. This strategy uses no information whatsoever, and even ignores the number of ships present at the lock. The first lockage is assumed to start at $t = 0$. When ship handling times are present, the lockage duration is adjusted accordingly. Ship waiting time is weighted by its priority value if required.

Move upon arrival (MA): The lock will not move unless a ship arrives or a ship is already waiting. Whenever a ship arrives on either side, the lock will immediately move and handle this ship (potentially by first performing an empty lockage so that a ship on the opposite side can enter). When given knowledge concerning the ships that have arrived so far while having no future arrival information, this may be the most straightforward way to operate the lock. Capacity, handling times and weights are again straightforward to take into account.

Wait until threshold (WUT): For this strategy, the lock is expected to move only when a certain number of ships is reached on either side of

the lock. We choose to linearly decrease this threshold value to zero over a constant interval I after each lockage. This guarantees that the lock will move at least once every $T + I$ time units. Note that this ensures that this heuristic always generates a feasible solution. If the threshold would not decrease over time, the last ship may not pass the lock if the threshold value is not met. Also note that, by choice of both the threshold value and I, this heuristic can be seen as a generalization of the CUD as well as the MA heuristic. The WUT results reported below assume an interval I of $T/2$, and a threshold of 2. Higher threshold values quickly decreased the performance for the instances used below. Capacity and handling times are again easy to include. When taking priority values into account, it makes sense to adjust the threshold value accordingly. We now consider the weighted number of ships waiting for service, and the original threshold value is multiplied by the mean of all ship priority values.

The heuristics above use no information about future arrivals. On real-world waterways however, ships should typically announce their arrival at a lock some time in advance. For example, as mentioned in Section 1.3.1, all cargo ships on the Rhine and Danube rivers are expected to communicate their time of arrival at least two hours before reaching a lock. More information can thus be expected to be available to the lockmaster. We present two additional heuristics below that incorporate some of this information in the decision process.

Minimum unavoidable increase (MUI): Assume that the lock is in a given position, say p, at a certain time t. A decision has to be made whether to move the lock immediately or to keep waiting for more ships. When the lock does not move, we assume that it waits at least until the next arrival at that position p. If no arrival occurs at position p within $2T$ time units, we restrict ourselves to this $2T$ horizon to limit the 'look-ahead' time. Thus, we assume the lock will leave at an arrival time of a ship, say t', with $t \leq t' \leq t + 2T$. Given t', we determine the *unavoidable increase* U_{wait} as the waiting time of all ships arriving at p, that were already present at time t; and of those ships that arrive in the interval $(t, t']$. In addition, all ships waiting at the opposite side $1 - p$, need to wait an additional T time units, compared to ships arriving at p, for the lock to arrive. This justifies the notion of 'unavoidable' waiting time; an increase in total waiting time at least as large as U_{wait} is incurred regardless of any future decisions. Alternatively, when the lock moves immediately, we apply a similar strategy. Once the lock starts to move, it cannot arrive back in its original position for a period of at least $2T$. We let U_{move} be equal to the unavoidable waiting time of all ships on the opposite

side as before, as well as the waiting time of all ships at the original position that can not be handled until the earliest time when the lock may return. The heuristic will now decide to wait when $U_{\text{wait}} < U_{\text{move}}$, or move immediately otherwise. This decision is repeated when the lock arrives at its new position, or at the end of the waiting time. Ship priorities and lock capacity can be incorporated without any problem. Some care must be taken when including handling times in the model, as the waiting time of ships positioned at the opposite site of the lock will then increase beyond T. Similarly, the earliest return time for the lock when determining U_{move} may be larger than $2T$. If capacity is included, it is assumed optimal to handle as many ships as the lock allows while handling ships with the shortest handling time first. All other aspects of the decision procedure remain unchanged.

Look-ahead 2T (LA2T): Like the previous heuristic, this procedure also makes use of future arrival information. The LA2T heuristic again decides whether or not to move the lock at certain decision times. However, we now base this decision on the optimum solution for the look-ahead interval. We assume that all arrival information is known for the next $2T$ time units and we create a new subproblem over this $2T$ horizon, also including all unhandled ships that are waiting at the decision time. This subproblem is then solved with the exact solution procedure outlined in Sections 2.3 and 2.5 while ensuring that the initial position of the lock is fixed. If the lock moves immediately in the solution to the subproblem, the heuristic now decides to move immediately in the original problem. If not, the lock waits until the next decision point, which corresponds to the arrival of a ship at the lock's position, or whenever a new ship is added to the $2T$ look-ahead period. Lock capacity as well as ship priorities are easy to add. Handling times cannot be included since an efficient procedure to find an exact solution is not available, as argued in Section 2.5.3. The look-ahead interval can be chosen arbitrarily large to improve the performance of this heuristic. However, repeated application of the exact procedure quickly increases the LA2T computation time beyond that of the other heuristics. For small look-ahead intervals such as $2T$, the exact procedure does find a solution quickly since the solution graph has very few edges in general.

For those problem settings where the first-come first-served assumption (i.e. Property 2.4) does not hold, each of the heuristics requires a sequencing rule to decide on the order in which the ships are handled. This is the case for the setting where capacity, priority, and handling times are considered simultaneously. We apply a greedy strategy in this case. Intuitively, one

expects that ships with high priority and/or short handling times should be handled first. A decision rule could be to order waiting ships by decreasing value of P/HT, where P is the priority value and HT the handling time. For our set of instances however, it turns out that handling ships by decreasing priority leads to significantly better performance. Many other greedy policies can be used, but are not considered below. For the setting with all extensions, we report results for the decreasing priority policy.

2.7 Computational study

Is it worth the effort to solve instances of the lockmaster's problem to optimality? Or, are heuristics sufficient to achieve near-optimal solutions? In this section we answer this question by performing computational experiments. We consider the basic problem setting as well as a number of extensions and compare the optimal solution to the heuristics described in Section 2.6. Each of the presented heuristics will be applied to all problem settings where possible. Note that the solution value for the presented heuristics depends on the initial position of the lock, whereas it does not for the exact solution; we compensate for this effect by obtaining a heuristic solution for both starting positions and reporting the best of both solutions.

2.7.1 Instances and problem setting

As far as we are aware, the only publicly available instances are maintained by Verstichel and Vanden Berghe (2009). We report results for these instances in Section 2.7.3. As these instances do not contain all information needed for the extensions we cover here, we also generate new instances for the experiments below, simulating a realistic arrival process. Since ship arrivals are independent, a Poisson arrival process is to be expected. The time between subsequent arrivals is then distributed according to an exponential distribution, since the 'memoryless' property should be satisfied. That is, the arrival probability for a ship is unrelated to the elapsed time since the last arrival. To obtain integral arrival times, we use a geometric distribution, which can be be seen as the discrete analogue of the continuous exponential distribution. The time unit is one minute. All instances consider a time horizon of 24 hours. The lockage duration is assumed to be 30 minutes, a typical value similar to that reported by Smith et al. (2009) for 'single' lock operations. The only remaining parameter to

decide upon is the parameter p that specifies the geometric distribution. This parameter corresponds to the arrival probability as follows: if we let X denote a random variable representing the number of ships that arrive at any given time in the instance, the following holds: $P(X \geq n) = p^n$ for all $n \in \mathbb{N}$. Each generated instance can thus be considered as a random realization of a 'typical day' where the arrival probability is specified by the value of p. We report here the results averaged over 25 random instances with equal p. Rijkswaterstaat has kindly provided a dataset with arrival information for the three parallel locks of Terneuzen, the Netherlands. Regression on the interarrival times shows that the geometric distribution assumed above corresponds well to reality. Furthermore, the arrival probability of $p = 0.1$ used in the simulations below reflects the arrival process observed at the locks of Terneuzen. In addition to the arrival times, each arriving ship is also assigned a priority value and a handling time for the extensions below. We consider the following problem settings:

1. **Basic**: The basic settings described in Section 2.3. This considerably simplified problem serves as the starting point for all cases below. Further, it acts as a reference to compare the relative performance of the different solution methods.

2. **Capacity**: The extension covered in Section 2.5.1. We assume that the number of ships in the lock is limited for any lockage. We arbitrarily set the bound equal to 3 for all capacitated instances below. Separate simulations, with capacity values of 6 and 12 respectively, indicating that the optimal solution is rarely restricted by higher capacity bounds. For this reason, results for the larger capacity bounds are omitted.

3. **Weights**: The extension considered in Section 2.5.2. All ships are given a priority value which is used as a weight factor in the objective function. Lock capacity is not limited in this case. The generated instances each have ship priority values chosen randomly from $\{1, 2, 3\}$ with equal probability.

4. **Handling times**: The extension considered in Section 2.5.3. All ships are given a handling time. For any lockage the lockage time is increased by the sum of the handling times of all ships present in the lock. For the instances below, handling time values are chosen randomly from $\{0, 2, 5\}$ with equal probability. Ship priorities and

p = 0.0333		basic	capacity	weights	handling	cap + W + HT
exact (min.)	OPT	727.3	793.8	1381.4	X	X
heuristic (%)	CUD	201.6%	195.8%	214.0%	1555.6	3296.1
	MA	192.3%	187.8%	204.2%	1557.6	3357.5
	WUT	212.6%	210.1%	214.4%	1649.7	3353.9
	MUI	115.2%	128.4%	110.3%	866.0	1982.4
	LA2T	107.2%	109.4%	104.6%	X	X

Table 2.1: Results for different problem settings. Each value represents the average of 25 instances simulating the same arrival process. The arrival parameter equals 1/30.

lock capacity are not considered. Note that finding the optimum solution is NP-hard as shown in Section 2.5.3.

To allow the fairest comparison of solution performance, the instances used for each of the extensions have arrival times identical to the corresponding instances in the basic setting.

2.7.2 Results

We summarize the results of all simulations in Tables 2.1 to 2.3. Each table shows values obtained by averaging over 25 instances for each heuristic under different problem settings and for a specific arrival probability (see Section 2.7.1). We do not separately list the required computation time as minimizing the solution time was not our main priority. Nevertheless, the exact solution is obtained in less than a second on average, except for the bounded capacity setting, where the increased graph complexity increases the computation time up to a 10 minutes average on a 3.4 GHz machine with 4GB RAM. Exact results are reported in minutes of total waiting time. Heuristic values are shown as a percentage relative to the corresponding exact solution where possible, and as total waiting time in minutes where the exact solution is not available. All generated instances and results for each individual instance are available online (`https://perswww.kuleuven.be/~u0086328/lockmasterdata.html`).

We see from the tables that the straightforward heuristics (CUD, MA, WUT) perform significantly worse than their look-ahead counterparts, though performance improves as the arrival probability increases. The WUT heuristic only realizes very small improvements over MA in some cases; we expect that there may in fact be certain conditions when such

p = 0.0666		basic	capacity	weights	handling	cap + W + HT
exact (min.)	OPT	1884.9	3768.8	3644.0	X	X
heuristic (%)	CUD	152.3%	130.4%	158.5%	3524.6	11972.6
	MA	153.1%	132.3%	159.2%	3516.6	12049.2
	WUT	156.3%	131.6%	158.7%	3472.3	11787.3
	MUI	115.7%	151.7%	114.7%	2397.2	12379.3
	LA2T	105.8%	107.6%	104.9%	X	X

Table 2.2: Results for different problem settings. Each value represents the average of 25 instances simulating the same arrival process. The arrival parameter equals 1/15.

p = 0.1		basic	capacity	weights	handling	cap + W + HT
exact (min.)	OPT	3361.3	21592.2	6474.6	X	X
heuristic (%)	CUD	137.1%	109.7%	141.1%	6166.0	60722.2
	MA	134.8%	108.4%	139.6%	6199.2	60003.0
	WUT	134.0%	107.4%	139.2%	6246.9	59995.4
	MUI	118.5%	138.9%	118.3%	5134.1	94393.9
	LA2T	105.3%	103.7%	105.3%	X	X

Table 2.3: Results for different problem settings. Each value represents the average of 25 instances simulating the same arrival process. The arrival parameter equals 1/10.

a threshold policy is beneficial. Especially when capacity is considered limited, moving a lock while it is not 'sufficiently' full may introduce unnecessary waiting times in future lockages.

When comparing the columns for the capacity simulations, it is clear that the lock quickly becomes heavily congested for higher values of p, as reflected in the large difference between optimal values for the capacity and basic settings. The optimality gap induced by all heuristics except for MUI tends to decrease as the arrival rate increases. This behaviour is to be expected as for a congested lock with a high arrival rate, the optimal decision would be to keep moving continuously up and down taking as many ships as possible. This decision also follows from each of the heuristics, except for MUI, which sometimes decides to wait for the next ship while moving is the better option. This tendency is most obvious when capacity is bounded, as the MUI heuristic does not explicitly consider capacity limits in its decision. However even when capacity is not an issue, the same tendency arises, to a minor extent, for the other problem settings. MUI is the only heuristic where performances decrease as the arrival rate increases.

LA2T clearly has the best heuristic performance overall. Making use of future arrival information clearly pays off. This appears to be the best available strategy, provided that the optimal solution to sub-problems can be found. The smaller size of sub-problems may also allow an exact MIP solution in reasonable time, even for settings that are known to be NP-hard. This suggests using this heuristic as a 'rolling horizon' strategy for large instances where the exact solution is infeasible. Increasing the look-ahead horizon will further improve the performance of this heuristic, though it should be mentioned that computation time likely becomes a restraining issue as the horizon increases. For our instances, the LA2T computation was at least an order of magnitude faster than for the exact solution, and frequently the difference was even larger. For one of the larger instances in the basic setting however, the exact solution for the entire problem was actually found faster, albeit only slightly, than the heuristic result.

2.7.3 Results for the instances by Verstichel and Vanden Berghe

Results for the instances maintained by Verstichel and Vanden Berghe (2009) are presented in Tables 2.4 to 2.7. We note that the lock and ship dimensions provided in these instances are ignored, since our approach

Instance	OPT	CUD	MA	WUT	MUI	LA2T
P_10_20_0.3	429	134%	125%	134%	116%	100%
P_10_20_0.5	400	138%	133%	140%	100%	108%
P_15_20_0.3	456	130%	117%	114%	153%	104%
P_15_20_0.5	385	145%	114%	140%	166%	103%
P_30_20_0.3	208	230%	171%	327%	145%	116%
P_30_20_0.5	261	190%	188%	194%	113%	119%
P_5_20_0.3	327	131%	149%	133%	100%	100%
P_5_20_0.5	428	116%	125%	118%	117%	106%
P_10_100_0.3	2281	131%	127%	131%	133%	109%
P_10_100_0.5	2084	134%	129%	134%	127%	104%
P_15_100_0.3	1993	142%	138%	147%	137%	106%
P_15_100_0.5	2002	146%	144%	132%	112%	105%
P_30_100_0.3	1238	247%	235%	242%	157%	110%
P_30_100_0.5	1334	202%	195%	235%	121%	111%
P_5_100_0.3	2438	128%	110%	128%	207%	118%
P_5_100_0.5	2434	118%	126%	119%	152%	109%
P_10_1000_0.3	5160	612%	536%	556%	466%	391%
P_10_1000_0.5	6195	527%	485%	471%	386%	327%
P_15_1000_0.3	5131	613%	598%	655%	340%	303%
P_15_1000_0.5	12023	285%	266%	241%	158%	142%
P_30_1000_0.3	9276	349%	323%	337%	124%	119%
P_30_1000_0.5	10123	352%	289%	307%	118%	113%
P_5_1000_0.3	10555	280%	278%	280%	395%	251%
P_5_1000_0.5	4873	597%	592%	597%	716%	544%

Table 2.4: Exact solution and heuristic performance for the 'Poisson' instances for the basic problem setting with unbounded lock capacity.

does not decide on the placement within the chamber. In contrast, we use the methods described in Sections 2.3 and 2.5.1 to obtain an optimum solution with respect to the provided arrival times. The lockage time T is chosen equal to 30. Relative weights and handling times are not included in these datasets. A subset of the instances has interarrival times which follow a Poisson distribution; other instances have times between arrivals following a uniform distribution. We note that interarrival times distributed according to a Poisson distribution do not reflect a Poisson arrival process, where an exponential distribution would be expected. Results for these instances are reported in Tables 2.4 and 2.6. Results for these instances are given in Tables 2.5 and 2.7. We do not report averages for the values in the tables as each of the instances simulates an arrival process with different parameters. We refer to the original paper by Verstichel and Vanden Berghe (2009) for details on the instance parameters. For Tables 2.6 and 2.7, where capacity is considered, we ignore the ship size and assume a capacity bound equal to 3 ships.

Instance	OPT	CUD	MA	WUT	MUI	LA2T
R_10_20_0.3	393	113%	134%	113%	123%	100%
R_10_20_0.5	421	124%	114%	124%	108%	100%
R_15_20_0.3	414	125%	125%	142%	114%	100%
R_15_20_0.5	368	144%	133%	161%	132%	112%
R_30_20_0.3	362	145%	151%	176%	111%	113%
R_30_20_0.5	350	166%	166%	138%	132%	100%
R_5_20_0.3	458	129%	134%	129%	120%	108%
R_5_20_0.5	444	127%	141%	129%	133%	109%
R_10_100_0.3	2234	124%	131%	124%	133%	108%
R_10_100_0.5	2168	129%	141%	129%	129%	106%
R_15_100_0.3	1832	156%	163%	159%	133%	103%
R_15_100_0.5	1815	168%	145%	154%	117%	104%
R_30_100_0.3	1533	187%	179%	212%	108%	106%
R_30_100_0.5	1181	224%	207%	248%	126%	104%
R_5_100_0.3	2568	110%	106%	110%	169%	104%
R_5_100_0.5	2424	118%	112%	118%	113%	111%
R_10_1000_0.3	11181	270%	266%	256%	207%	173%
R_10_1000_0.5	7225	451%	374%	416%	320%	288%
R_15_1000_0.3	13685	225%	221%	221%	117%	108%
R_15_1000_0.5	11577	293%	253%	251%	156%	145%
R_30_1000_0.3	3914	841%	727%	805%	280%	280%
R_30_1000_0.5	8040	443%	360%	385%	134%	134%
R_5_1000_0.3	19062	151%	149%	151%	255%	141%
R_5_1000_0.5	6174	469%	476%	469%	577%	442%

Table 2.5: Exact solution and heuristic performance for the 'Uniform' instances for the basic problem setting with unbounded lock capacity.

Instance	OPT	CUD	MA	WUT	MUI	LA2T
P_10_20_0.3	696	126%	129%	126%	184%	100%
P_10_20_0.5	577	117%	124%	121%	102%	100%
P_15_20_0.3	704	144%	118%	119%	229%	113%
P_15_20_0.5	530	117%	128%	147%	275%	100%
P_30_20_0.3	208	230%	171%	327%	145%	116%
P_30_20_0.5	261	190%	188%	194%	113%	119%
P_5_20_0.3	767	157%	165%	160%	146%	123%
P_5_20_0.5	736	108%	130%	108%	167%	102%
P_10_100_0.3	14647	103%	109%	103%	131%	101%
P_10_100_0.5	5204	103%	123%	103%	201%	101%
P_15_100_0.3	2852	135%	132%	133%	280%	103%
P_15_100_0.5	2359	139%	140%	146%	138%	111%
P_30_100_0.3	1238	247%	235%	242%	157%	110%
P_30_100_0.5	1334	202%	195%	235%	121%	111%
P_5_100_0.3	42271	106%	103%	106%	123%	103%
P_5_100_0.5	26060	106%	104%	106%	116%	100%

Table 2.6: Exact solution and heuristic performance for the 'Poisson' instances for the limited capacity setting with bound equal to 3.

Instance	OPT	CUD	MA	WUT	MUI	LA2T
R_10_20_0.3	806	107%	125%	107%	175%	100%
R_10_20_0.5	665	114%	135%	114%	183%	100%
R_15_20_0.3	758	124%	108%	115%	169%	108%
R_15_20_0.5	368	144%	133%	161%	309%	122%
R_30_20_0.3	415	156%	142%	156%	112%	100%
R_30_20_0.5	350	166%	166%	138%	149%	100%
R_5_20_0.3	1373	113%	115%	113%	149%	115%
R_5_20_0.5	844	159%	152%	159%	195%	121%
R_10_100_0.3	20490	109%	103%	109%	117%	107%
R_10_100_0.5	9130	105%	104%	105%	169%	102%
R_15_100_0.3	8123	107%	110%	115%	173%	119%
R_15_100_0.5	2392	152%	132%	145%	241%	108%
R_30_100_0.3	1543	186%	177%	211%	118%	113%
R_30_100_0.5	1314	211%	198%	230%	136%	104%
R_5_100_0.3	36030	103%	105%	103%	132%	103%
R_5_100_0.5	29384	105%	102%	105%	113%	100%

Table 2.7: Exact solution and heuristic performance for the 'Uniform' instances for the limited capacity setting with bound equal to 3.

2.8 Conclusion

In this chapter, we introduced 'the lockmaster's problem', a simplified problem setting that represents the core problem underlying any lock scheduling setting that involves batch processing of ships and the scheduling of lockage operations. This problem is closely linked to known batch scheduling problems. The problem can be solved in polynomial time; we were able to construct a graph such that applying a shortest path algorithm solves the lockmaster's problem to optimality. We investigated a number of problem extensions based on practical considerations such as the chamber capacity and ship-dependent handling times. We provided hardness results for these generalizations, in addition to describing a modified solution method to obtain an optimum solution when a first-come first-served policy is enforced. Finally, computational experiments confirmed that the exact algorithm outperforms a number of basic heuristics.

One possible direction for future research is to focus on the batch scheduling aspect of the problem. For example, it may be interesting to extend the single-lock setting to a machine scheduling problem with job compatibilities and jobs of more than two types. Throughout this chapter, it is assumed that all input data are known initially. An extension where this is no longer the case could introduce some uncertainty about the input parameters. A different extension relates to the arrival times, which may

only be known in a limited time window that updates as time progresses. These generalizations are discussed further in Chapter 6, since they are also relevant for the problem settings discussed in the remaining chapters.

Chapter 3

Computational complexity of scheduling locks in sequence

Although the results obtained in Chapter 2 can be useful to obtain an operational schedule, it rarely occurs that there is only a single lock along the entire length of a well-travelled waterway. For example, the two most economically important waterways in Europe are the Danube and the Rhine. The Rhine-Main-Danube Canal and the Main river, that connect the Danube to the Rhine, respectively have 16 and 34 locks along this connection. Clearly, ships travelling along these waterways will encounter many locks during their itinerary, and individually scheduling each of the locks may not yield an overall optimum solution with respect to the waiting time objective, see also Section 4.5 for an illustration.

In this chapter, we consider the problem of obtaining an integrated schedule for multiple locks arranged in a sequence. The general definition of this problem setting is provided in Section 3.1. Section 3.2 gives an overview of related literature. Our results focus on the nature of exact algorithms for different versions of the problem. More concrete, we show that the existence of polynomial-time algorithms must be considered unlikely, even for the case of two identical locks and all ships travelling in the same direction. Indeed, we show that this problem is strongly NP-hard in Section 3.3. In Section 3.4, we show that the case of identical ships also remains strongly NP-hard for an arbitrary number of locks. Section 3.5

introduces a class of solutions which satisfy particular properties: so-called synchronised solutions. Section 3.5.3 describes a polynomial-time algorithm for a uni-directional variant of the latter problem, thus narrowing the gap between known easy and hard problem settings.

The first main contribution of this chapter is to prove strong NP-hardness for the problem of scheduling single-chamber locks in sequence. This is, to the best of our knowledge, the first proof where the complexity of the problem is not due to the packing of ships within chambers or due to the chamber assignment. (For a NP-hardness proof related to the chamber assignment, see Chapter 5.) We show that the problem is NP-hard in two fundamentally different special cases. The first special case is a setting with identical locks and uni-directional travel, where ships may be non-identical. The second special case is a setting with identical ships. The claim that bi-directional travel contributes fundamentally to the complexity in the latter setting is reinforced by providing a polynomial-time algorithm for a uni-directional variant with identical ships. When we use the phrase "identical locks", we mean that all locks have an equal lockage duration and all locks have equal capacity, i.e. $T_l = T$ and $C_l = C$ for all $l \in \mathcal{L}$. When we use the phrase "identical ships", we mean that all ships travel at equal speed, i.e. $V_s = V$ for all $s \in \mathcal{S}$.

The second contribution of this chapter is to investigate the existence of so-called synchronised solutions. We show that there exists an optimum solution which is synchronised in two special cases of our problem. The first special case is a setting with two identical locks and bi-directional travel; the second special case is a setting with multiple identical locks and uni-directional travel. Additionally, we consider different generalizations of the two-lock setting and show that these results do not apply to a number of extended problem settings.

3.1 Problem definition

Given is an ordered set of single-chamber locks $\mathcal{L} = \{1, \ldots, L\}$. Since each lock consists of a single chamber, we simplify our notation as follows. Each lock $l \in \mathcal{L}$ has a lockage duration T_l and a capacity C_l. Throughout this chapter we will, unless otherwise mentioned, consider locks with unbounded capacity; that is, the setting where $C_l = +\infty$ for all $l \in \mathcal{L}$. Recall that, for all $l \in \mathcal{L} \setminus \{L\}$, S_l specifies the distance between locks l and $l+1$. Given the set of ships $\mathcal{S}$, recall that the arrival time of a ship $s \in \mathcal{S}$ is given by A_s. Recall from Section 1.3.1 that, in general, a ship need not enter the

system of locks at the first or the last lock. The arrival and departure locks of a ship s are denoted by L_s^A and L_s^D respectively. Throughout this chapter we assume that each ship $s \in \mathcal{S}$ travels at a fixed speed V_s.

The problem consists of finding a feasible schedule with minimum total waiting time. We denote this problem by SLS, short for Sequential Lock Scheduling. In Section 3.3, we show that solving SLS is strongly NP-hard by proving that the corresponding decision problem is strongly NP-complete. This decision problem, denoted by dec-SLS, is formulated as follows: "given an instance of SLS, does there exist a feasible schedule with a total waiting time of at most W?"

3.2 Relation to literature

To the best of our knowledge, no results on the complexity of scheduling locks in sequence exist in the literature. A known result is that the problem becomes strongly NP-hard as soon as the placement of ships within a chamber is considered. See e.g. Verstichel et al. (2014a) for a description of the ship placement problem. A problem related to our problem, also in the context of waterways, is the scheduling of bi-directional traffic along a narrow river or canal, where a limited number of wider segments is available for the crossing of ships that travel in opposite directions, or for the overtaking of ships that travel in the same direction. The complexity of the problem to minimize total waiting time for this setting is settled by Disser et al. (2015). We note that the results obtained by Disser et al. (2015) do not immediately extend to our setting for the scheduling of locks. One attempt to connect the two problems is to see the narrow canal sections in their problem setting as locks, and the widened sections as the distance separating adjacent locks. But then, a notable difference is that in the lock setting, each lock must return to its initial position in between the processing of two ships that travel in the same direction, whereas a narrow canal is immediately available for ships travelling in either direction. Other differences are that, in our lock setting, each ship is allowed to overtake any other ship while both ships are travelling, and that the distance between locks in fact enforces a minimum amount of required waiting time for the widened sections.

A large body of related literature also exists on the scheduling of a single-track railway line with a limited number of segments or stations where overtaking is allowed. An early paper on this topic was presented by Frank (1966); since then, many papers have studied integer programming

models and heuristics for this problem setting. While the single-track railway scheduling problem is similar to the sequential locks scheduling problem, a number of differences appear. First, a lock may serve multiple ships simultaneously in a single lockage. Secondly, in the lock scheduling problem, travel time is associated with each section between locks; in the railway setting such a section corresponds to a single track segment, which requires a train to stand still and incur waiting time in order to allow overtaking. Gafarov et al. (2015) prove NP-hardness for such a single-track railway problem. In their problem setting, each of the trains has a due date, so that the overtaking of trains may in fact be required, and the hardness result follows in part from the underlying sequencing problem.

A third problem setting that relates to our SLS setting is the well-known flow shop problem, where a set of jobs is given and each job is to be executed on a set of machines. Typical for a flow shop is that each of the jobs must be executed by each of the machines in a pre-defined order which is identical for all jobs. Clearly, the jobs correspond to ships arriving over time, and the machines correspond to the different locks. The natural ordering of locks is immediately reflected in the order in which the jobs must be executed on the machines. See, for example, Emmons and Vairaktarakis (2013) for an overview of the many results on the flow shop problem and its variants.

One aspect in which the SLS problem is clearly more general is the bi-directional traffic: in terms of the flow shop problem, this corresponds to the existence of jobs which must be processed in 'reverse order'. The complexity of such an extension falls naturally between that of the original flow shop problem and that of the job shop problem, where the processing order of each job is independently specified. The SLS problem, however, is more specific than the flow shop problem in terms of the job processing time; whereas this processing time may be arbitrarily chosen for each combination of job and machine in the flow shop problem, the processing time on any lock (i.e. machine) is clearly identical for each of the ships (i.e. jobs) in SLS. Other notable differences between SLS and the flow shop problem are the batch processing of jobs with the same direction, the requirement that processing alternates between batches with different directions, and the fact that not all ships need to pass each of the locks in the SLS problem. The travel time needed on the waterway segments in between the locks can be modelled in the flow shop setting as a transportation delay, also referred to as transfer lag, between the different machines. Brucker et al. (2004) give an overview of flow shop scheduling problems with transportation delays and introduced a number of new complexity results. Notice that the

transportation delay is independent of the job if the speed of all ships is identical, whereas general transportation delays can represent any arbitrary fixed speed for each ship. When the distance between locks is considered to be zero or, equivalently, when the speed of all ships is large enough for the travel to be considered equal to zero, transportation delays can be ignored.

Results on the generalization of the flow shop problem to two directions for the processing order are scarce. One setting where this variant is considered, is presented by Zhao et al. (2009); they consider a sequence of operations for a container loading and unloading problem. A bi-directional flow shop problem with additional constraints on the processing order is described, and the authors present integer programming formulations as well as a heuristic procedure based on a relaxation of the original problem.

While the complexity of SLS does not immediately follow from results in the literature, we are able to denote several special cases of the SLS problem using the standard notation for scheduling problems; for a description of this notation, see e.g. Graham et al. (1979). Consider the uni-directional variant of SLS. The distinction between job families then disappears, and the act of returning the lock to its original position before processing the next ship can be considered as a fixed setup time which can be anticipated, i.e. executed before a job is available at a machine. We show in Section 3.3 that SLS is strongly NP-hard, even for two identical machines and with arbitrary capacity. A related result for a two-machine flow shop was obtained by Brucker et al. (2004), who showed that problem $F2|p_{i,j} = 1, r_j, t_j|\sum C_j$ is strongly NP-hard. Here, r_j represent the release date of jobs, which is easily seen to correspond to the arrival times of the ships, and t_j represents a job-dependent transportation delay, which reflects the ship-dependent travel time between the locks. Notice that in the flow shop setting, in contrast to problem SLS, multiple jobs cannot be processed simultaneously in a single batch.

3.3 A hardness result for two identical locks

We prove strong NP-hardness for a uni-directional special case of SLS. In fact, the reduction we provide below will imply strong NP-hardness for a significantly more restricted setting. The precise result is as follows.

Theorem 3.1. *Problem dec-SLS is strongly NP-complete, even for two identical locks, all ships travelling in the same direction, and when each ship must be processed by both locks.*

Proof. Inspired by a proof by Disser et al. (2015), we start our reduction from MAX CUT, which consists of answering the following question: given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, does there exist a cut consisting of at least K edges? This problem is shown to be NP-hard, see Garey et al. (1976). Notice that we consider the unweighted case, sometimes referred to as SIMPLE MAX CUT.

For a given instance of MAX CUT, we describe a corresponding instance of the decision version of SLS. We will then argue that solving dec-SLS for this instance corresponds to deciding the question of MAX CUT. We first turn G into a directed graph by choosing some ordering of the vertices in V, and next orienting every edge from the vertex with smaller index to the vertex with larger index.

The instance of dec-SLS is as follows. There are two identical locks. The lockage time of each of these locks equals 1 (i.e. $T_1 = T_2 = 1$), the capacity of each lock is infinite (i.e. $C_1 = C_2 = \infty$), and the distance between the two locks equals 1 (i.e. $S_1 = 1$). Lock 1 is positioned upstream of lock 2. The set of ships consists of a total of $2nm + 2n + 2m$ ships. All ships travel in the downstream direction and must first be served by the first lock and subsequently by the second lock. For ease of exposition, we distinguish two types of ships: *vertex ships* and *edge ships*. Note that each ship must be served by both locks and that the speed of the ships need not be identical, as will be described in what follows.

We now specify the arrival times A_s for each $s \in \mathcal{S}$. Recall that we have imposed an arbitrary order on V: let $V = \{1, \ldots, n\}$. For each vertex $v \in V$, we have $m + 1$ vertex ships arriving at time $5(v - 1)$ and $m + 1$ vertex ships arriving at time $5(v-1)+1$. We thus have a total of $2nm+2n$ vertex ships. We say that the time interval $[5(v - 1), 5v)$ on the first lock is the period corresponding to vertex v on the first lock, and the time interval $[5n + 5(v - 1), 5n + 5v)$ is the period corresponding to vertex v on the second lock. The speed of each vertex ship equals $1/(5n-1)$, i.e. each vertex ship needs $5n - 1$ time units to travel the distance between the two locks.

In addition, there are two edge ships for each $(v_i, v_j) \in E$ (with $v_i < v_j$): one ship arriving at time $5(v_i-1)$, and one ship arriving at time $5(v_i-1)+1$. We will refer to these arrivals as the *first ship* and *second ship* corresponding to edge (v_i, v_j) respectively. Clearly, we have $2m$ edge ships in total. The speed of the first ship for each such edge equals $1/(5(n+v_j-v_i))$, while the speed of the second ship equals $1/(5(n+v_j-v_i)-2)$.

The question is: does there exist a solution with total waiting time at most $W \equiv nm + n + 3m - 2K$? This completes the description of the

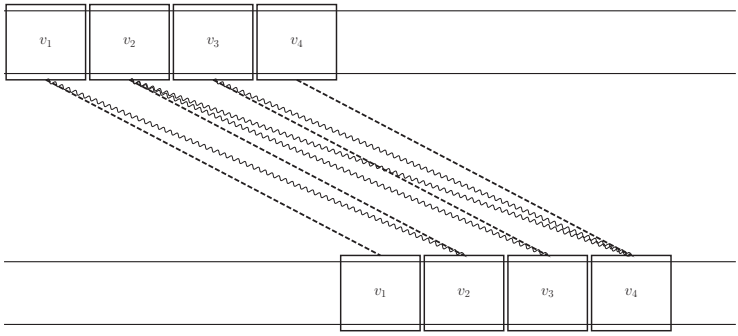


Figure 3.1: Overview of a constructed instance of dec-SLS. The blocks represent periods corresponding to the different vertices. Dashed lines represent vertex ships, while waved lines represent edge ships.

instance of dec-SLS. An overview of the constructed instance of dec-SLS is shown in Figure 3.1. A detailed illustration of the vertex ships and edge ships follows in the remainder of this section.

Before we argue the equivalence between a yes-instance of MAX CUT and a yes-instance of dec-SLS, let us first explicitly describe two possible ways of serving the ships arriving in a given period. Figure 3.2 illustrates two possible ways of serving the ships arriving in a period corresponding to some vertex v on the first lock. In the first option, a downwards lockage starts at times $5(v-1)$ and $5(v-1)+2$, and an upwards lockage starts at time $5(v-1)+1$. Ships arriving at time $5(v-1)$ can thus be served immediately upon their arrival, while ships arriving at time $5(v-1)+1$ incur a waiting time of 1 time unit. We refer to this way of serving the ships in this period as *option 1*. A second way to serve these ships is to schedule downward lockages starting at time $5(v-1)+1$ and $5(v-1)+3$, and an upwards lockage starting at time $5(v-1)+2$. Notice that ships arriving at time $5(v-1)$ then incur at least one unit of waiting time, while ships arriving at time $5(v-1)+1$ can be served immediately upon their arrival. We refer to this way of serving the ships that arrive in this period as *option 2*.

Given these two options for serving the arrivals at lock 1, we can define two similar options for each period on lock 2. Let option 1 on the second lock consist of downwards lockages starting at times $5n+5(v-1)$ and $5n+5(v-1)+2$, and an upwards lockage at time $5n+5(v-1)+1$; let option 2 on the second lock consist of downwards lockages starting at

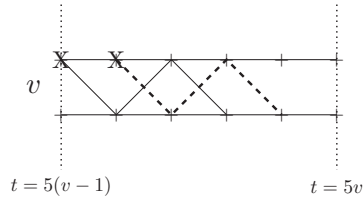


Figure 3.2: Illustration of option 1 (solid lines) and option 2 (dashed lines) to serve the ships arriving in a period on the first lock.

times $5n + 5(v-1) + 1$ and $5n + 5(v-1) + 3$, and an upwards lockage starting at time $5n + 5(v-1) + 2$.

Notice that, on either lock, options in different periods are independent of each other since there is enough time to reposition the lock in any appropriate state after having transferred the ships of an earlier period. For example, selecting option 1 in the period $[5, 10)$ does not prevent us from selecting either option 1 or option 2 in the periods $[0, 5)$ or $[10, 15)$, on either of the locks.

We now argue the equivalence between a yes-instance of MAX CUT, and a yes-instance of dec-SLS. Suppose that there exists a cut in the graph G with at least K edges; let the corresponding partition of the node-set V be indicated by V_1 and V_2. We build the following solution for the instance of dec-SLS. If vertex $v \in V_1$, then we use option 1 for the two periods corresponding to vertex v; if vertex $v \in V_2$, then we use option 2 for the two periods corresponding to vertex v. We claim that the solution that arises when (i) each ship arriving at the first lock enters the first possible lockage; next, (ii) immediately travels to the second lock after leaving the first lock; and finally (iii) enters the first possible lockage of the second lock, is a solution with total waiting time bounded by W.

We first revisit the vertex ships. Observe that in each period on lock 1, no matter whether option 1 or option 2 is used, one half of the arriving vertex ships in that period has no waiting time, and the other half incurs a waiting time of 1 time unit. This accounts for a total waiting time of $n(m+1)$ for the vertex ships, at the first lock. Recall that these ships require a travel time of $5n - 1$ in between the two locks. Suppose that option 1 is selected for the period corresponding to v on lock 1. This means that at time $5(v-1) + 1$ and at time $5(v-1) + 3$, there are $m + 1$ ships leaving lock 1 and travelling towards lock 2. It follows that these ships arrive on the upstream position of lock 2 at times $5n + 5(v-1)$ and

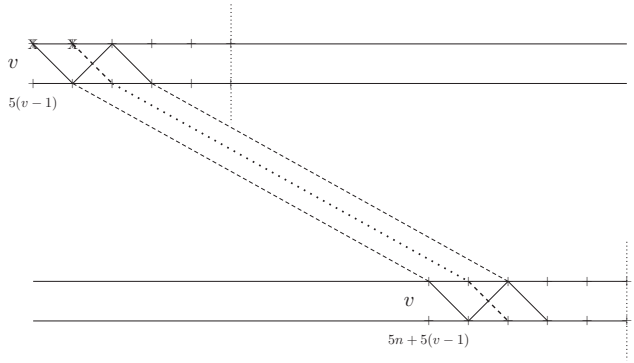


Figure 3.3: Illustration of the vertex ships in a period corresponding to a vertex v. Each 'X' marks the arrival of $m + 1$ vertex ships.

$5n + 5(v - 1) + 2$ respectively. Observe that this period on lock 2 also corresponds to vertex v and hence uses option 1. Thus, these ships do not incur any waiting time at lock 2. In the case that $v \in V_1$, we thus have a waiting time of $m + 1$ for the vertex ships corresponding to v. In the case that $v \in V_2$, a similar argument can be made. Indeed, the $m + 1$ ships arriving at time $5(v - 1)$ then each incur a waiting time of 1 time unit at the first lock; all vertex ships arriving in the period corresponding to v leave the first lock at time $5(v - 1) + 2$ and arrive at the upstream side of the second lock at time $5n + 5(v - 1) + 1$. Since option 2 was selected for this period, these ships do not incur any additional waiting time at lock 2. It follows that the total waiting time due to vertex ships equals $n(m + 1)$. An illustration of this construction is provided in Figure 3.3.

We now look at the edge ships. Consider an edge $(v_i, v_j) \in E$ with $v_i < v_j$. Recall that the first ship corresponding to this edge requires a travel time of $5(n + v_j - v_i)$, and the second ship requires a travel time of $5(n+v_j-v_i)-2$. For convenience, let $t_i = 5(v_i-1)$ and $t_j = 5n+5(v_j-1)$, so that t_i and t_j are the starting time of the period corresponding to v_i on lock 1 and the starting time of the period corresponding to v_j on lock 2, respectively. We now have the following four cases to consider. Figure 3.4 illustrates these cases graphically.

- *Case 1:* option 1 is used for the period corresponding to v_i on lock 1 and option 1 is used for the period corresponding to v_j on lock 2. Then, the first ship enters lock 1 at its arrival time t_i, leaves lock

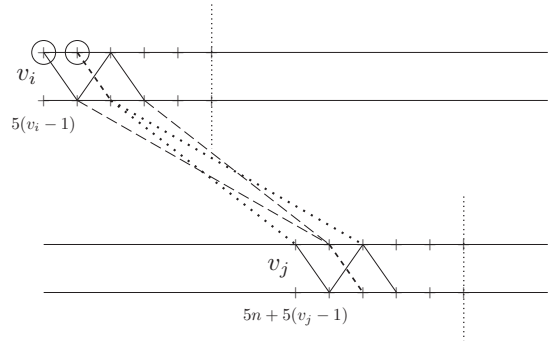


Figure 3.4: Illustration of the edge ships in a period corresponding to an edge (v_i, v_j). Each circle marks the arrival of a single edge ship.

1 at $t_i + 1$, and due to its speed, arrives at lock 2 at time $t_j + 1$, where it has to wait 1 time unit in order to be served. The second ship waits 1 time unit in front of lock 1, leaves lock 1 at $t_i + 3$, and arrives at lock 2 at time $t_j + 1$. Hence, the second ship also has to wait 1 time unit in front of lock 2. The total waiting time for these two ships in this case equals 3 time units.

- *Case 2:* option 1 is used for period v_i on lock 1 and option 2 is used for period v_j on lock 2. Then, the first ship leaves lock 1 at $t_i + 1$ and arrives at lock 2 at time $t_j + 1$. The second ship incurs one unit of waiting time at lock 1, leaves lock 1 at time $t_i + 3$ and arrives at lock 2 also at time $t_j + 1$. Hence, neither ship incurs any additional waiting time at lock 2; the total waiting time for these two ships in this case equals 1 time unit.

- *Case 3:* option 2 is used for period v_i on lock 1, and option 1 is used for period v_j on lock 2. Then, both the first and second ship enter lock 1 at time $t_i + 1$ and leave lock 1 at $t_i + 2$. Due to their speeds, the first ship arrives at lock 2 at time $t_j + 2$, while the second ship arrives at lock 2 at time t_j. It follows that neither ship incurs any additional waiting time at lock 2; the total waiting time for these two ships in this case equals 1 time unit.

- *Case 4:* option 2 is used for period v_i on lock 1, and option 2 is used for period v_j on lock 2. Then, both the first and second ship enter lock 1 at time $t_i + 1$ and leave lock 1 at $t_i + 2$. Due to their speeds,

the first ship arrives at lock 2 at time $t_j + 2$, while the second ship arrives at lock 2 at time t_j. Both ships incur an additional waiting time of 1 time unit. The total waiting time for these two ships in this case equals 3 time units.

We conclude this case analysis by observing that if the same option is selected for a period v_i on lock 1 and a period v_j on lock 2 that correspond to an edge $(v_i, v_j) \in E$, there is a waiting time of 3 time units corresponding to this edge. If the two selected options for the periods corresponding to this edge differ, there is a waiting time of 1 time unit.

Since the cut contains K edges, we infer that the edge ships have a total waiting time of $K + 3(m - K) = 3m - 2K$. Indeed, observe that if edge (v_i, v_j) is in the cut, i.e. if $v_i \in V_1$ and $v_j \in V_2$, the two options used for periods v_i and v_j differ, resulting in a waiting time of 1 corresponding to this edge; otherwise, there is a waiting time of 3. Hence, the total waiting time for all ships equals $n(m+1)+3m-2K = W$. A yes-instance of MAX CUT thus gives rise to a yes-instance of dec-SLS.

Consider now a solution to an instance of SLS with a total waiting time of at most W. First, we argue that we can assume that such a solution is so-called *sensible*. We say that a solution to dec-SLS is sensible if

- *Condition 1:* each ship enters the first downwards lockage that occurs at or after its arrival at a lock,
- *Condition 2:* in each period on lock 1 and on lock 2, either option 1 or option 2 is used.

We argue that we can restrict ourselves to sensible solutions only.

Lemma 3.1. *For any feasible solution to SLS with total waiting time W', there exists a sensible solution to SLS with total waiting time at most W'.*

Proof. It is easily argued that Condition 1 can be enforced without increasing the total waiting time. Indeed, given any solution that does not satisfy Condition 1, it is obvious that each ship for which an earlier lockage is available can be immediately reassigned to that earlier lockage; this does not increase the total waiting time. We note that this also holds when the capacity of each lock is bounded: each ship then enters the first non-full lockage at or after its arrival at a lock.

To see that Condition 2 can be guaranteed, observe that there exists an optimal solution where each lockage of a lock l either (i) starts at a moment in time where some ship that is served by the lockage arrives

at this lock l, or (ii) starts immediately upon the completion time of a preceding lockage of this lock l. (Recall that this observation is also made for a single lock in Chapter 2, see Property 2.1.) Consider now a period corresponding to some vertex v on the first lock. It follows that we can restrict ourselves to solutions where the first downwards lockage starts at time $5(v-1)$ or at time $5(v-1)+1$. This corresponds to either option 1 or option 2.

As a result, on the second lock, ships arrive only at times t, $t+1$, or $t+2$, where t denotes the starting time of a period corresponding to some vertex $v \in V$. Thus, the same argument can be repeated to see that selecting either option 1 or option 2 yields minimum waiting time for any given period on the second lock. Since this holds independently for each period on each lock, the claim follows. □

We may now assume that the given solution for dec-SLS is a sensible solution with a total waiting time bounded by W. Let us argue that the instance of MAX-CUT is then a yes-instance. Since our solution for dec-SLS is a sensible solution, it follows that, in each period, either option 1 or option 2 is used. We again consider the waiting time of the two types of ships. For each vertex $v \in V$, observe that there are $2(m+1)$ vertex ships arriving in the period corresponding to v on the first lock. These ships incur a total waiting time of $m+1$ if both periods corresponding to v use the same option, and a total waiting time of $3(m+1)$ if the two periods use different options. For the edge ships, recall that the two ships corresponding to an edge (v_i, v_j) incur a total waiting time of 1 if different options are used for the period corresponding to v_i on the first lock and the period corresponding to v_j on the second lock; if the same option is used for these periods, the total waiting time equals 3 time units. Observe that, since there are m edges and n periods where vertex ships arrive, a total waiting time of $n(m+1)+m$ cannot be avoided.

We claim that the two periods corresponding to any given vertex $v \in V$ must use the same option. We argue by contradiction. Recall that the total waiting time must equal at least $n(m+1)+m$ and that an additional waiting time of $2(m+1)$ is incurred for every vertex v for which the two corresponding periods are scheduled with different options. Assume that there is a single vertex for which this is the case. It follows that the total waiting time is then at least $n(m+1)+m+(2m+1) > nm+n+3m-2K = W$. This contradicts that our solution for dec-SLS has a waiting time of at most W. It follows that for each vertex $v \in V$, the two corresponding periods are scheduled using the same option. We can

conclude that in any sensible schedule with a total waiting time no greater than W, the total waiting time equals $n(m+1)+m$ plus an additional waiting time of 2 time units for every edge $(v_i, v_j) \in E$ with $v_i < v_j$ where the periods corresponding to v_i are scheduled in the same state as the periods corresponding to v_j.

Finally, we construct a solution to MAX CUT by assigning a vertex v to V_1 (V_2) if option 1 (option 2) is used for the two periods corresponding to vertex v. Then, since the solution to dec-SLS has a total waiting time no greater than $W = nm + n + m + 2(m - K)$, it follows that there are at least K pairs of vertices (v_i, v_j) with $v_i < v_j$ such that the period corresponding to v_i on lock 1 is scheduled with a different option than the period corresponding to v_j on lock 2. Indeed, if only $K-1$ pairs use different options, the total waiting time equals at least $nm+n+(K-1)+3(m-(K-1)) = nm+n+3m-2K+2$. By construction, there are thus at least K pairs of vertices (v_i, v_j) with $v_i < v_j$ such that exactly one of these vertices is in V_1 and the other is in V_2. Thus, there are at least K edges in the resulting cut in G. A yes-instance of dec-SLS thus gives rise to a yes-instance of MAX CUT, which completes our reduction. □

As a remark, we note that the construction of this reduction can be modified so that each lock has unit capacity and all arrivals occur at distinct times. This can be achieved by extending the length of each period corresponding to a vertex and spreading all simultaneous arrivals out over time. Notice that Lemma 3.1 also holds in this more general setting. We omit a formal description of this proof.

3.4 A hardness result for identical ships

We now provide an alternative NP-hardness proof for the SLS problem with identical ships, i.e. where the speed of all ships is the same. Note that in the reduction outlined below, ships travel in both directions. A crucial difference with the reduction provided in Section 3.3 is that, here, the number of locks is not bounded by a constant independent of the input. Furthermore, ships need not be served by each of the locks; as described in Section 1.3.1, we specify an arrival lock L_s^A and a departure lock L_s^D for all ships $s \in \mathcal{S}$.

The general outline of the reduction is inspired by a reduction for a problem involving bi-directional traffic on a path, described by Disser

et al. (2015). This setting, however, does not correspond exactly to the lock scheduling setting, as mentioned in Section 3.2. The wording and presentation of the proof in this section also resemble the proof of Theorem 3.1.

Theorem 3.2. *Problem dec-SLS is strongly NP-complete, even for identical ships and identical locks with unbounded capacity.*

Proof. We again start our reduction from MAX CUT; recall that an instance of MAX CUT consists of a graph $G = (V, E)$ and a non-negative integer K. To aid the exposition of the reduction, we first provide a general overview of the reduction before describing all details. The total number of locks in the SLS instance is not bounded by a constant, although we argue below that this number is bounded by $O(m)$. On the locks, we describe periods that correspond to the vertices in the instance of MAX CUT. A crucial part of the construction is that, on each lock that contains these periods, the order of the vertices corresponding to the periods is permuted. As in Section 3.3, we will argue that there are only two sensible scheduling options for each period; the option that is chosen in a given period reflects the assigned partition in the corresponding instance of MAX CUT.

Figure 3.5 illustrates a simplified overview of the fundamental part of the construction: a set of periods on a sequence of locks that represents the existence of an edge. In the figure, this is shown for an edge (v_1, v_4). Each square in the figure represents a period on one of the locks; each period corresponds to a vertex. We say that two periods are *adjacent* if they occupy consecutive time intervals on the same lock. We say that two periods are *diagonally adjacent* if they occupy consecutive time intervals on two consecutive odd-numbered locks. Note that in Figure 3.5, only the odd-numbered locks are shown. The role of the even-numbered locks will be specified in the detailed description of the construction. The main components of the construction are (i) sets of ships, represented in the figure by dashed lines, which ensure that two periods occupying the same time interval on different locks correspond to the same vertex, (ii) sets of ships, represented in the figure by waved lines, which ensure that the vertices to which two adjacent periods correspond are interchanged from one lock to the next, and (iii) sets of ships, represented in the figure by dotted lines, which correspond to the existence of an edge between the vertices corresponding to two diagonally adjacent periods. In what follows we argue that the constructed instance of SLS corresponds to a given instance of MAX CUT, and provide a detailed description of these three components.

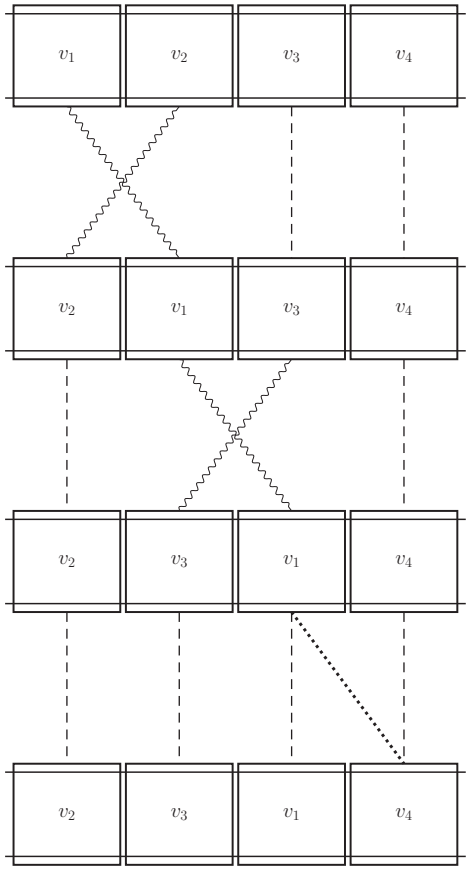


Figure 3.5: Illustration of the construction for an edge (v_1, v_4). Only the odd-numbered locks are shown. Each box represents a period corresponding to a vertex. Dashed lines connect periods corresponding to the same vertex on the same time interval; waved lines connect adjacent periods for which the corresponding vertex is interchanged; dotted line models the existence of an edge between diagonally adjacent periods.

To define the set of ships in the SLS instance, we distinguish ships of different types. All ships travel at unit speed, i.e. each ship traverses one unit of distance per unit of time. The two types of ships are as follows:

1. ships of *type 1*, arriving on the upstream position of a specified lock l and travelling towards the downstream position of this lock l. Each type 1 ship thus only traverses a single lock. We have $L_s^A = L_s^D = l$ and $s \in \mathcal{D}$ for all ships s of type 1.

2. ships of *type 2*, arriving at a specified lock l; each ship of this type may be upstream-travelling or downstream-travelling, and requires processing by three locks. For a ship $s \in \mathcal{S}$ of type 2, we thus have either $L_s^A = l$, $L_s^D = l - 2$ and $s \in \mathcal{D}$, or $L_s^A = l$, $L_s^D = l + 2$ and $s \in \mathcal{U}$. In what follows, the travel direction and characteristics will be distinguished as needed.

To construct an instance of dec-SLS, we use an algorithmic description. This algorithm runs a procedure for each edge $(v_i, v_j) \in E$ where $v_i < v_j$. We initialize by specifying the first lock and next we use the procedure described below. Each odd-numbered lock in the instance has n periods, each corresponding to a vertex in V. A period consists of a time interval on a lock l; the i'th period spans a time interval $[24(i-1), 24i)$. For convenience we define $t_i = 24(i-1)$, which equals the starting time of the i'th period on each of the locks. Note that, as illustrated in Figure 3.5, the i'th period on a lock l does not necessarily correspond to the vertex v_i, and the order in which periods correspond to vertices need not be the same on different locks. The lockage duration equals $T_l = 2$ and the travel distance equals $S_l = 6$ for each lock in the instance. The locks in $\mathcal{L}$ remain ordered from upstream to downstream, i.e. as new locks are added to $\mathcal{L}$ throughout the procedure, they are added to the downstream end.

Initialization

We start with a single lock: $\mathcal{L} = \{1\}$. At this lock 1 we have, for each vertex $v \in V$, $m + 1$ arrivals of type 1 at each of the times $24(v-1)$, $24(v-1)+2$, $\ldots$, $24(v-1)+18$. We define the i'th period on lock 1, i.e. the period on the interval $[24(i-1), 24i)$, to be the period corresponding to vertex i, for all $i \in \{1, \ldots, n\}$. For convenience, we also define the value $l^* = 1$, representing the last lock in $\mathcal{L}$ at each step in the construction procedure where new locks are added to $\mathcal{L}$. Figure 3.6 illustrates the arrivals of type 1 in such a period.

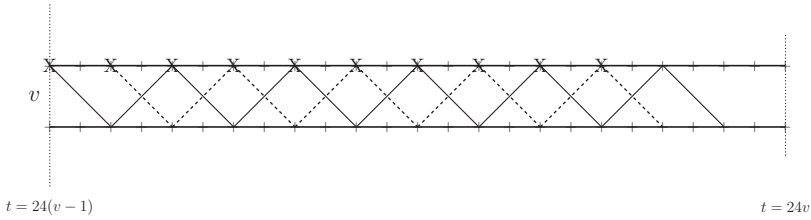


Figure 3.6: Illustration of a period on some lock, corresponding to a vertex. Each 'X' marks the arrival of $m + 1$ type 1 ships. Solid lines correspond to option 1; dashed lines correspond to option 2.

As in the proof of Section 3.3, we first highlight two possible ways to schedule a lock to serve the arrivals that arrive within a period on some lock. Let t be the starting time of the period. *Option 1* consists of scheduling a series of consecutive lockages starting with a downwards lockage at time t and ending with a downwards lockage that starts at time $t + 20$; *option 2* consists of scheduling a series of consecutive lockages starting with a downwards lockage at time $t + 2$ and ending with a downwards lockage that starts at time $t + 18$. Observe that if either of these options is used, the ships of type 1 incur a total waiting time of $10(m + 1)$. Also notice that options in different periods are independent of each other since there is enough time to reposition the lock to either the upstream or the downstream position after having transferred the ships of an earlier period. For example, selecting option 1 in the period $[24, 48)$ does not prevent us from selecting either option 1 or option 2 in the periods $[0, 24)$ or $[48, 72)$.

For each edge $(v_i, v_j) \in E$, we now describe a procedure that specifies a set of locks, and arrivals at these locks, to be added to the partial instance. This procedure is ran for each edge once, and after the final edge the instance is complete.

Procedure repeated for each edge

Consider some edge $(v_i, v_j) \in E$ with $v_i < v_j$. Let i and j be the periods corresponding to v_i and v_j on lock l^*, respectively. We assume that $t_i < t_j$; if this does not hold, we simply swap i and j below. The procedure is as follows.

1. While the periods corresponding to vertices v_i and v_j on lock l^* are not adjacent, repeat

(a) We add two new locks: let $\mathcal{L} \leftarrow \mathcal{L} \cup \{l^* + 1, l^* + 2\}$.

(b) On lock $l^* + 2$, we have periods corresponding to vertices: for each $i \in \{1, \ldots, n\}$, we add $m + 1$ arrivals of type 1 at each of the times t_i, $t_i + 2$, $\ldots$, $t_i + 18$.

(c) Let v_k be the vertex to which the period $[t_i + 24, t_i + 48)$ on lock l^* corresponds; note that this is the period following the period corresponding to vertex v_i on lock l^*. Clearly, $v_k \neq v_i$ and $v_k \neq v_j$. We say that, on lock $l^* + 2$, vertex v_k corresponds to period $[t_i, t_i + 24)$, and vertex v_i corresponds to period $[t_i + 24, t_i + 48)$. Notice that, compared to lock l^*, the vertices corresponding to these two periods are interchanged on lock $l^* + 2$. All other vertex-period correspondences remain equal to those on lock l^*.

(d) On lock l^*, we add $m + 1$ arrivals of type 2 at each of the times $t_i + 16$ and $t_i + 18$. These ships travel in the downstream direction and are thus served by locks l^*, $l^* + 1$, and $l^* + 2$. Additionally, on lock $l^* + 2$, we add $m + 1$ arrivals of type 2 at each of the times $t_i + 10$ and $t_i + 12$. These ships travel in the upstream direction and are thus served by locks $l^* + 2$, $l^* + 1$, and l^*. For convenience, we will refer to the ships added in this step as ships of type 2a. Notice that these ships are added in the periods for which the vertex-period correspondence changes from lock l^* to lock $l^* + 2$.

(e) On all remaining periods on lock l^*, i.e. all periods $[t_k, t_k + 24)$ for which $t_k \neq t_i$ and $t_k \neq t_i + 24$, we add $m + 1$ arrivals of type 2 at both time t_k and time $t_k + 2$. For convenience, we will refer to the ships added in this step as ships of type 2b. Notice that these ships are added in the periods for which the vertex-period correspondence remains the same from lock l^* to lock $l^* + 2$.

(f) We update l^* so that it again refers to the latest added lock in the instance. That is, we set $l^* \leftarrow l^* + 2$. Further, we redefine i and j to be the periods corresponding to v_i and v_j on the new last lock l^*; we adjust t_i and t_j accordingly.

2. Observe that the periods corresponding to v_i and v_j are now adjacent on lock l^*. We add two additional locks: let $\mathcal{L} = \mathcal{L} \cup \{l^* + 1, l^* + 2\}$. On lock $l^* + 2$, we again have periods corresponding to vertices: for

each $i \in \{1, \ldots, n\}$, we add $m + 1$ arrivals of type 1 at each of the times t_i, $t_i + 2$, ..., $t_i + 18$.

3. For each $i \in \{1, \ldots, n\}$, we add $m + 1$ arrivals of type 2 at each of the times t_k and $t_k + 2$. We refer to the ships added in this steps as ships of type 2b. Notice that the vertex-period correspondence on lock l^* and lock l^* is the same for all periods.

4. Finally, we add arrivals corresponding to the edge (v_i, v_j). We add a single arrival of type 2 on lock l^* at each of the times $t_i + 12$ and $t_i + 14$. We refer to these arrivals as ships of type 2c. Additionally, we add $m + 1$ arrivals of type 1 on lock $l^* + 1$ at each of the times $t_i + 21$ and time $t_i + 23$.

This concludes the formal description of an instance of SLS corresponding to a given instance of MAX CUT. Figure 3.5 shows the structure of an example where the procedure is applied for an edge (v_1, v_4). In the figure, detailed arrival times and the even-numbered locks are not shown. Upon completing the construction of this instance, let N_1, N_{2a}, and N_{2b} equal the total number of ships of type 1, type $2a$, and type $2b$ respectively. Note that both the total number of ships and the number of locks are polynomial in the size of the original instance G. Indeed, for each edge in E, at most n interchange operations are performed: we extend the construction with $O(n)$ locks for each edge. On each lock, at most $O(nm)$ ships are added. The total number of locks is thus bounded by $O(nm)$; the total number of ships is bounded by $O(n^2m^2)$. The decision question to be answered in the corresponding instance of dec-SLS is the following. "Does there exist a solution with a total waiting time of at most $W \equiv N_1 + N_{2a} + N_{2b} + 10m - 4K$?"

Correspondence of MAX CUT to dec-SLS

We first state the foundations of the argument which shows the correspondence between the given instance of MAX CUT and the constructed instance of dec-SLS. Notice that, on each odd-numbered lock, we have a period corresponding to each vertex. We will argue that we can restrict ourselves to solutions of SLS where all periods are scheduled using either option 1 or option 2 and, moreover, all periods corresponding to the same vertex are scheduled using the same option. The selected option then indicates one of two possible partitions of V to which a vertex is assigned, thus defining a cut in the given graph G. We now proceed by providing a

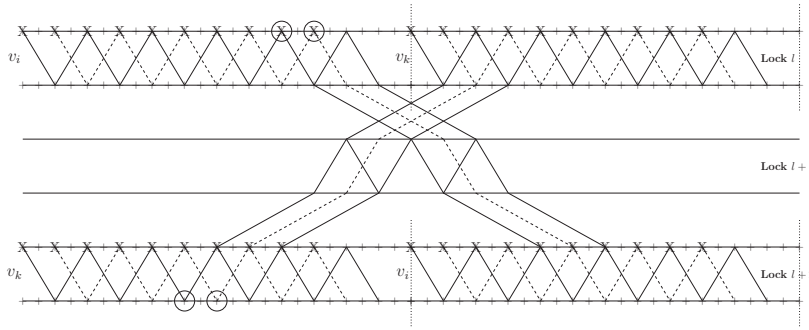


Figure 3.7: Construction corresponding to the interchange of two adjacent periods. Each 'X' marks the arrival of $m+1$ ships of type 1. Each circle marks the arrival of $m+1$ ships of type 2a.

detailed overview of the different arrivals added throughout the construction, and the waiting time incurred by these arrivals depending on the chosen option for the different periods.

Figure 3.6 gives a detailed representation of a period $[24(v-1), 24v)$ on some lock, corresponding to some vertex. Recall that, in each such period, we have $10(m+1)$ arrivals of type 1 and that, if either option 1 or option 2 is used in this period, the total waiting time incurred by these ships equals $10(m+1)$.

Step 1(d) adds ships of type 2a, corresponding to the waved lines in Figure 3.5; these arrivals are added where the vertex-period correspondence of two adjacent periods is interchanged from some lock l to lock $l+2$. A detailed representation, depicting all arrivals in the corresponding periods is shown in Figure 3.7. Observe that if either option 1 or option 2 is chosen for each period, although ships travelling in opposite direction cross in between locks l and $l+1$, no lockages overlap regardless of the chosen option for the periods. Further observe that, if the same option is chosen for the two periods corresponding to v_i, a total waiting time of $2(m+1)$ is incurred by the $2(m+1)$ ships of type 2a arriving on lock l; similarly, if the same option is chosen for the two periods corresponding to v_k, a total waiting time of $2(m+1)$ is incurred by the $2(m+1)$ ships of type 2a arriving on lock $l+2$. If different options are used for either the two periods corresponding to v_i (repectively v_k), the total waiting time for the ships of type 2a arriving on lock l (respectively $l+2$) equals at least $6(m+1)$.

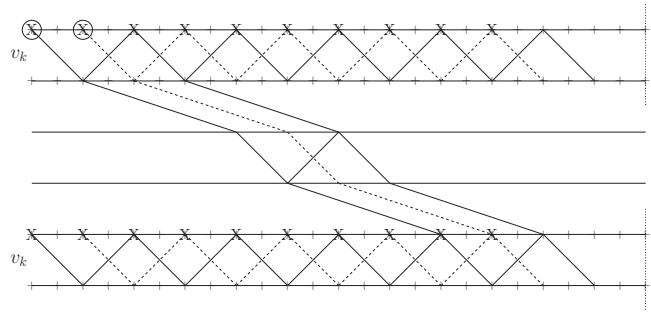


Figure 3.8: Construction corresponding to two periods that occupy the same time interval and correspond to the same vertex. Each 'X' marks the arrival of $m+1$ type 1 ships. Each circle marks the arrival of $m+1$ type 2b ships.

Step 1(e) and step 3 add ships of type 2b, corresponding to the dashed lines in Figure 3.5; these ships travel between a period on a lock l and a period on lock $l+2$ that occupy the same time interval and correspond to the same vertex. Figure 3.8 shows a detailed representation depicting all arrivals in the corresponding periods. Observe that if the same option is chosen for the two periods corresponding to v_k, a total waiting time of $2(m+1)$ is incurred by the ships of type 2b; if different options are used for the two periods, the waiting time equals at least $6(m+1)$.

Step 4 adds ships of type 2c, corresponding to the dotted lines in Figure 3.5. A detailed representation of this construction is shown in Figure 3.9. Observe that if two different options are used for the periods corresponding to v_i and v_j on lock l and $l+2$, the ships of type 2c incur a total waiting time of at least 6 time units. In contrast, if the same option is used for both periods, the total waiting time for the ships of type 2c equals at least 10 time units. Also notice that on the even-numbered lock $l+1$, in order to achieve this waiting time, a downwards lockage must be scheduled at times t_i+21 and t_i+25 if option 1 is selected for the period corresponding to v_i, and a downwards lockage must be scheduled at time t_i+23 if option 2 is selected for the period corresponding to v_i. The arrivals of type 1 on the intermediate lock then incur a total waiting time of $2(m+1)$. Note that whenever these arrivals of type 2c are added to a period, this period also has arrivals of type 2b as described by step 5 of the construction. There is, however, no overlap in the trajectory of

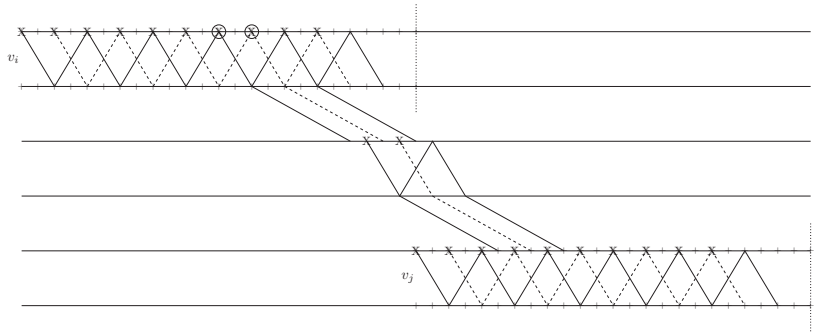


Figure 3.9: Construction corresponding to an edge (v_i, v_j). Each 'X' marks the arrival of $m + 1$ type 1 ships. Each circle marks the arrival of a single type 2c ship.

the ships of types 2b and 2c, nor is there an overlap between the lockages serving these ships in Figures 3.8 and 3.9.

We are now ready to argue that a cut in graph G containing at least K edges exists if and only if a solution exists for the instance of dec-SLS with a waiting time of at most W.

$\Rightarrow$ Assume that there exists a cut in G that contains at least K edges; the corresponding partition of the vertices is indicated by V_1 and V_2. We build the following solution for the instance of dec-SLS. If vertex $v \in V_1$, then we use option 1 for all periods corresponding to vertex v; if vertex $v \in V_2$, then we use option 2 for all periods corresponding to vertex v; each ship enters the first available lockage corresponding to its direction of travel, and the lockages for all even-numbered locks are scheduled such that ships of type 2c incur a waiting time of 1 time unit at these locks, as illustrated in Figure 3.9. We claim that the resulting waiting time of all ships is bounded by W.

We first identify the total waiting time for the ships of types 1, 2a, and 2b. Recall that either option 1 or option 2 is chosen for each period in the instance.

- The total waiting time for ships of type 1 equals the total number of type 1 arrivals. Indeed, arrivals of type 1 are added either (i) in a period corresponding to a vertex (Figure 3.6), or (ii) on an even-numbered lock where two adjacent periods are connected to model an edge in E (Figure 3.9). In both of these cases, the total

waiting time incurred by type 1 ships equals the number of type 1 arrivals.

- The total waiting time for ships of type 2a equals the total number of type 2a arrivals. Indeed, arrivals of type 2a are added only where two periods occupy the same time interval and correspond to the same vertex (Figure 3.8). As a result, the same option is chosen for these two periods, and the total waiting time incurred by type 2a ships equals the number of type 2a arrivals.

- The total waiting time for ships of type 2b equals the total number of type 2b arrivals. Indeed, arrivals of type 2b are added only where the vertices to which two periods correspond on a lock l are interchanged on lock $l+2$ (Figure 3.7). As a result, the same option is chosen for each pair of periods corresponding to the same vertex, and the total waiting time incurred by type 2b ships equals the number of type 2b arrivals.

For these ships, this yields a total waiting time of $N_1 + N_{2a} + N_{2b}$.

Now consider the ships of type 2c. Observe that for each edge $(v_i, v_j) \in E$, there are exactly two arrivals of type 2c. If edge (v_i, v_j) is in the cut, i.e. if $v_i \in V_1$ and $v_j \in V_2$ or vice versa, the options used for periods v_i and v_j differ on the locks traversed by these ships. This results in a waiting time of 6 corresponding to this edge; otherwise, there is a waiting time of 10. Since there exists a cut of K edges, we thus obtain a total waiting time of $6K + 10(m - K) = 10m - 4K$ for the ships of type 2c. The total waiting time for the corresponding solution of SLS equals $N_1 + N_{2a} + N_{2b} + 10m - 4K = W$. A yes-instance of MAX CUT thus gives rise to a yes-instance of dec-SLS.

$\Leftarrow$ Consider now a solution to an instance of SLS with a total waiting time of at most W. First, we argue that we can assume that such a solution is so-called sensible. We say that a solution is *sensible* if:

- *Condition 1:* after arriving at a lock, each ship enters the first available lockage corresponding to its direction of travel,

- *Condition 2:* in each period corresponding to a vertex, either option 1 or option 2 is used,

- *Condition 3:* each even-numbered lock is scheduled such that ships of type 2c traversing that lock incur a waiting time of 1.

We argue that we can restrict ourselves to considering sensible solutions only.

Lemma 3.2. *For any feasible solution to SLS with waiting time W', there exists a sensible solution to SLS with waiting time at most W'.*

Proof. Notice that Conditions 1 and 2 are identical to the definition a sensible solution used in Lemma 3.1 in the context of non-identical ships. The argument that these conditions can be guaranteed, without increasing the total waiting time of a solution, can be repeated from the proof of Lemma 3.1.

To see that Condition 3 can be enforced without increasing the total waiting time, we again make use of the fact that there is an optimum solution where, for each lock l, each lockage starts at a point in time where a ship arrives at lock l or follows immediately upon an earlier lockage of lock l. It is then clear that for the corresponding time t_i in the step in the construction where ships of type 2c are added, visualised in Figure 3.9, the type 1 ships that arrive at the even-numbered lock can be served either by (i) a downwards lockage starting at time $t_i + 20$ and a downwards lockage starting at time $t_i + 24$, (ii) a downwards lockage starting at time $t_i + 21$ and a downwards lockage starting at time $t_i + 25$, (iii) a single downwards lockage starting at time $t_i + 22$, or (iv) a single downwards lockage starting at time $t_i + 23$. Observe that the schedule of the even-numbered lock only determines the total waiting time of ships of type 1 arriving on the lock and the total waiting time of ships of type 2c. It can be seen that if option 1 is selected for the period corresponding to v_i, case (i) or (ii) must hold in any optimum solution; if option 2 is selected, case (iii) or (iv) must hold in any optimum solution. If case (i) or (iii) holds, it is not difficult to see that, by selecting option (ii) or (iv) respectively, the total waiting time for ships of type 1 reduces by $2(m + 1)$ time units, whereas it increases for the ships of type 2c by at most 6 time units. Clearly we may assume $m \geq 2$ since instances with $m = 1$ are trivial to solve. Case (ii) or case (iv) above must then hold in any optimum schedule; it follows that all ships of type 2c then incur a single unit of waiting time at an even-numbered lock. □

We may thus assume that the given solution for SLS is a sensible solution with a total waiting time bounded by W. We argue that the instance of MAX-CUT is then a yes-instance. We first claim that all periods corresponding to any given vertex $v \in V$ must use the same option. We argue by contradiction. Recall that the total waiting time in

any sensible schedule equals at least $N_1 + N_{2a} + N_{2b} + 6m$ and that an additional waiting time of $4(m+1)$ is incurred for every vertex v for which two corresponding periods are scheduled with different options. Assume that there is a single vertex for which this is the case. It follows that the total waiting time must be equal to at least $N_1 + N_{2a} + N_{2b} + 10m + 4 > N_1 + N_{2a} + N_{2b} + 10m - 4K = W$. This contradicts the fact that our solution for dec-SLS has a waiting time of at most W. Thus, all periods corresponding to some vertex $v \in V$ are scheduled using the same option. For any sensible schedule with a total waiting time no greater than W, it then follows that we have a solution where the total waiting time consists of:

1. $N_1 + N_{2a} + N_{2b}$, incurred by ships of type 1, type 2a, and type 2b,

2. a waiting time of 6 time units for every edge $(v_i, v_j) \in E$ where different options are chosen for the periods corresponding to v_i and v_j, incurred by ships of type 2c,

3. a waiting time of 10 time units for every edge $(v_i, v_j) \in E$ where the same option is chosen for the periods corresponding to v_i and v_j, incurred by the remaining ships of type 2c.

Given a solution to SLS with a total waiting time of at most $W = N_1 + N_{2a} + N_{2b} + 10m - 4K$, we construct a solution to MAX CUT by assigning a vertex v to V_1 (respectively V_2) if option 1 (option 2) is used for the periods corresponding to vertex v. It follows that there are at least K pairs of vertices (v_i, v_j) with $v_i < v_j$ such that the periods corresponding to v_i are scheduled with a different option than the periods corresponding to v_j. Indeed, if at most $K-1$ pairs use different options, the total waiting time must equal at least $N_1 + N_{2a} + N_{2b} + 6(K-1) + 10(m-(K-1)) > W$. By construction, there are thus at least K pairs of vertices (v_i, v_j) with $v_i < v_j$ such that exactly one of these vertices is in V_1 and the other is in V_2. Thus, there are at least K edges in the resulting cut in G. A yes-instance of dec-SLS thus gives rise to a yes-instance of MAX CUT, which completes our reduction. □

We remark here that the construction outlined above, like the proof described in Section 3.3, can be modified so that each lock has unit capacity and arrivals occur at distinct times. We omit a formal description for this modified setting.

3.5 Synchronised solutions

In this section, we introduce a class of solutions that posses a specific property: so-called synchronised solutions. We discuss two problem settings that are special cases of SLS, and show that in these settings, there exists an optimal solution which is synchronised. The first setting, covered in Section 3.5.1, is the special case where there are two identical locks with zero travel time, and where each ship must be processed by both locks. Furthermore, we show by means of different examples that the existence of an optimum solution which is synchronised does not extend to a number of generalizations of this two-lock setting. The second setting, covered in Section 3.5.3, considers uni-directional travel along a sequence of identical locks, where the arrival and departure locks of the ships are such that there is at least one 'common lock', i.e. there is at least one lock that must process each of the ships. Notice that, in this second setting, the ships need not all pass each of the locks, and travel time may be non-zero.

We begin by stating the definition of a *synchronised* solution.

Definition 3.1. *A solution to SLS is synchronised if each ship $s \in \mathcal{S}$ only incurs waiting time before entering a lockage at its arrival lock L_s^A.*

Observe that if a solution is synchronised, it follows that each ship $s \in \mathcal{U}$ (respectively $s \in \mathcal{D}$), after having been served by a lock $l < L_s^D$ ($l > L_s^D$), travels to lock $l+1$ ($l-1$) and immediately enters a lockage of lock $l+1$ ($l-1$) without incurring any waiting time. Figure 3.10 illustrates this definition.

Note that synchronised solutions correspond to so-called 'green waves' in traffic scheduling; for example, when scheduling a series of traffic lights along an important road, it makes sense to adjust the timing of the lights so that once a car meets a first green light, it can keep travelling at the indicated maximum speed and arrive at all following traffic lights without encountering any red lights. When considering bi-directional traffic and locks, a crucial difference with the scheduling of traffic lights appears: a green light simultaneously serves cars going in either direction, whereas a lock cannot.

3.5.1 Synchronised solutions for two locks

We consider a special case featuring two identical locks with infinite capacity. The travel distance between the locks is equal to zero (or, equivalently, the travel time is considered negligible) and each ship must pass both locks.

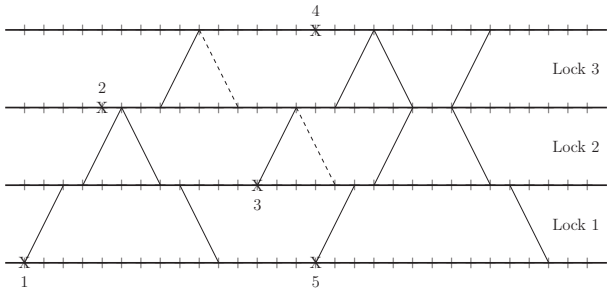


Figure 3.10: An example of a synchronised schedule. This example features 3 locks, with distances $S_1 = 1$, $S_2 = 2$. All odd-numbered ships are upstream-travelling and travel to lock 3, whereas even-numbered ships are downstream-travelling and travel to lock 1. Dashed lines correspond to empty lockages.

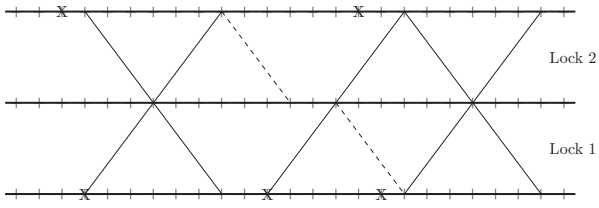


Figure 3.11: An example of a synchronised schedule featuring two identical locks. The distance between the locks is equal to zero. The dashed lines correspond to empty lockages.

We consider the bi-directional setting, i.e. downstream-travelling ships as well as upstream-travelling ships may be present. Figure 3.11 shows an instance and a feasible synchronised solution illustrating this special case of SLS. In the figure, the shown solution is also optimum. In fact, we show in this section that, for this problem setting, there always exists an optimum solution which is synchronised. Recall that in a synchronised solution, by definition, a ship $s \in \mathcal{S}$ incurs waiting time only before entering its arrival lock L_s^A.

Theorem 3.3. *For each instance of SLS consisting of two identical locks where the travel distance equals zero and where all ships must be served by both locks, there exists an optimum solution which is synchronised.*

Proof. We prove the theorem by arguing that an arbitrary optimum solution $\mathcal{O}$ can be transformed into a synchronised solution $\mathcal{O}'$ without increasing the total waiting time. Let $\mathcal{I}$ be an instance of the stated problem. We call the locks in this instance lock 1 and lock 2, with lock 2 positioned upstream of lock 1. Consider an arbitrary optimum solution $\mathcal{O}$ which is not synchronised. Let the total waiting time of this solution be equal to W.

We observe the lockages of lock 2 in solution $\mathcal{O}$. If the first lockage of lock 2, with starting time t_f, is an upwards lockage, we add a downwards lockage with starting time $t_f - T$ on lock 2 in solution $\mathcal{O}$. If the last lockage of lock 2, with starting time t_l, is a downwards lockage, we add an upwards lockage with starting time $t_l + T$ on lock 2 in solution $\mathcal{O}$. Clearly, the added lockages are empty and do not alter the total waiting time of the solution. Observe that the i'th lockage of lock 2 is then a downwards lockage if i is odd, and an upwards lockage if i is even. We now consider all odd-numbered lockages i of lock 2, and their subsequent lockage $i+1$. Let t_i and t_{i+1} be the starting time of the lockages i and $i+1$ respectively. We show how we can construct a synchronised solution $\mathcal{O}'$ while retaining optimality. Clearly, $t_{i+1} - (t_i + T) \geq 0$. We distinguish the following three cases for the starting times of lockages i and $i+1$ in solution $\mathcal{O}$.

- *Case 1:* $t_{i+1} - (t_i + T) = 0$. This situation is shown in Figure 3.12. We schedule the following lockages in solution $\mathcal{O}'$. On lock 2, we schedule a downwards lockage starting at time t_i and an upwards lockage starting at time t_{i+1}, i.e. we copy the lockages of lock 2 in $\mathcal{O}$; on lock 1, we schedule an upwards lockage starting at time $t_{i+1} - T$ and a downwards lockage starting at time $t_i + T$. Notice that scheduling these lockages does not occupy either lock outside of the interval $[t_i, t_{i+1} + T]$. Also notice that any ship served by lockage i in solution $\mathcal{O}'$ leaves lock 1 no later than in solution $\mathcal{O}$, and that any ship served by lockage $i+1$ in solution $\mathcal{O}'$ leaves lock 2 no later than in solution $\mathcal{O}$.

- *Case 2:* $0 < t_{i+1} - (t_i + T) < 2T$. This situation is shown in Figure 3.13. If lockage i in solution $\mathcal{O}$ is empty, we may reschedule it without increasing the total waiting time so that it starts at time $t_{i+1} - T$. If lockage $i+1$ in solution $\mathcal{O}$ is empty, we may reschedule it without increasing the total waiting time so that it starts at time $t_i + T$. If either of the above is the case then lockages i and $i+1$ are consecutive, and we schedule lockages on lock 1 as described in Case 1 above. Assume, thus, that both lockages i and $i+1$ are

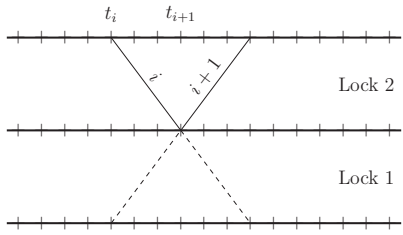


Figure 3.12: Visualisation of Case 1 in proving Theorem 3.3.

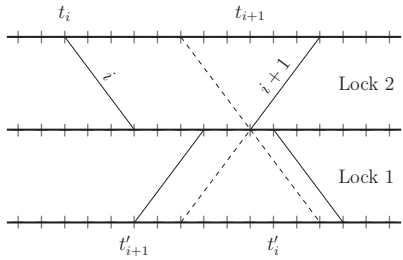


Figure 3.13: Visualisation of Case 2 in proving Theorem 3.3.

non-empty. Since each ship must be served by each lock, there clearly exist lockages on lock 1 serving the ships in lockages i and $i+1$. Let t'_{i+1} be the starting time of the latest upwards lockage of lock 1 in solution $\mathcal{O}$ with $t'_{i+1} + T \leq t_{i+1}$. If $t'_{i+1} + T < t_{i+1}$, it follows that lockage $i+1$ can be rescheduled to start earlier, at a time $t_{i+1} - \epsilon$ for some $\epsilon > 0$. Since lockage $i+1$ is non-empty, this operation reduces the total waiting time, contradicting the optimality of solution $\mathcal{O}$. Thus, $t'_{i+1} + T = t_{i+1}$. Let t'_i be the starting time of the earliest downwards lockage of lock 1 in solution $\mathcal{O}$ with $t'_i \geq t_i + T$. Notice that $t'_{i+1} < t'_i$. Indeed, t'_{i+1} and t'_i cannot coincide and if $t'_{i+1} > t'_i$ we have $t_{i+1} \geq t'_{i+1} + T \geq t'_i + 2T \geq t_i + 3T$, which contradicts our assumption that $t_{i+1} - (t_i + T) < 2T$. It then follows from $t'_i \geq t_{i+1}$ that in solution $\mathcal{O}$, lockage i can be rescheduled to start later, at time $t_{i+1} - T$ without increasing the total waiting time. Lockages i and $i+1$ are then consecutive, and we schedule lockages on lock 1 as described in Case 1 above.

- *Case 3:* $t_{i+1} - (t_i + T) \geq 2T$. This situation is shown in Figure 3.14.

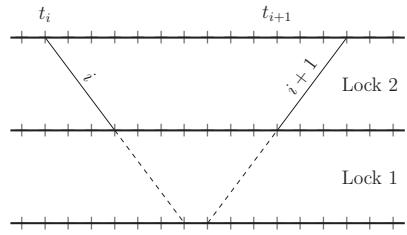


Figure 3.14: Visualisation of Case 3 in proving Theorem 3.3.

We schedule lockages in solution $\mathcal{O}'$ as follows. On lock 2, we schedule a downwards lockage starting at time t_i, and an upwards lockage starting at time t_{i+1}. On lock 1, we schedule a downwards lockage starting at time $t_i + T$ and an upwards lockage starting at time $t_{i+1} - T$. As in Case 1, it can be seen that scheduling these lockages does not occupy either lock outside of the interval $[t_i, t_{i+1} + T]$, and that any ship served by lockages i or $i+1$ leaves its departure lock no later than in solution $\mathcal{O}$.

For every pair of lockages $(i, i+1)$ with i odd, the solution $\mathcal{O}'$ consists of an interval $[t_i, t_{i+1} + T)$ containing either the structure in Figure 3.12 or Figure 3.14. It is easily verified that a solution consisting exclusively of such structures is synchronised. Furthermore, since the intervals containing these structures do not overlap, solution $\mathcal{O}'$ is feasible. Finally, since the total waiting time in solution $\mathcal{O}'$ is, by construction, not greater than the total waiting time of the solution $\mathcal{O}$, solution $\mathcal{O}'$ is also optimum. □

We note that the proof can be seen to hold in two slightly more general settings. If the setting from Theorem 3.3 is modified so that it features non-identical lockage durations T_1 and T_2, the proof remains valid if lock 2 is chosen to be equal to the lock with the largest lockage duration. Similarly, if the setting from Theorem 3.3 is modified so that it features arbitrary capacity bounds C_1 and C_2, the proof remains valid if lock 2 is chosen to be the lock with the smallest capacity. We point out, however, that the theorem does not hold the setting which features an arbitrary lockage duration as well as an arbitrary capacity for both locks, as will be shown in the following section.

Ships					Locks			
s	A_s	direction	L_s^A	L_s^D	l	T_l	C_l	S_l
1	0	$\mathcal{U}$	1	2	1	2	2	0
2	0	$\mathcal{U}$	1	2	2	1	1	-

Table 3.1: Instance for the example illustrating Observation 3.1.

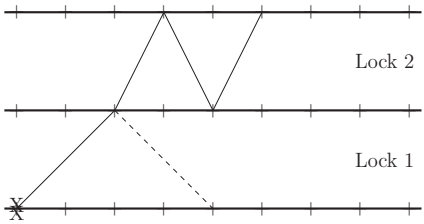


Figure 3.15: Feasible solution illustrating Observation 3.1.

3.5.2 Synchronised solutions in general

We show that, in general, there may not always exist an optimal solution that is synchronised. There are four conditions formulated in Theorem 3.3: (i) the locks are identical, (ii) there are two locks, (iii) the travel distance is equal to zero, and (iv) each ship is served by each lock, i.e. for all $s \in \mathcal{U}$: $L_s^A = 1$ and $L_s^D = L$, and for all $s \in \mathcal{D}$: $L_s^A = L$ and $L_s^D = 1$. Each of these conditions is necessary, as shown by the following examples.

Two arbitrary locks

The first generalization we consider is the setting where the two locks are not identical. Let T_1, T_2 and C_1, C_2 denote the lockage duration and the capacity of the two locks, respectively. Consider the instance described in Table 3.1 and the solution (which is not synchronised) shown in Figure 3.15. Clearly, the total waiting time in this solution equals 2 time units. Observe that in any optimum solution, lock 1 starts an upwards lockage at time 0 containing both ships, otherwise the total waiting time equals at least 4 time units. Then, since both ships cannot be served by lock 2 at time 2, there exists no feasible synchronised solution with a waiting time less than 4 time units. We can conclude the following.

Ships				
s	A_s	direction	L_s^A	L_s^D
1,2,3	0	$\mathcal{D}$	3	1
4	3	$\mathcal{U}$	1	3
5	7	$\mathcal{D}$	3	1
6,7,8	11	$\mathcal{U}$	1	3

Locks			
l	T_l	C_l	S_l
1	3	∞	0
2	3	∞	0
3	3	∞	-

Table 3.2: Instance for the example illustrating Observation 3.2.

Observation 3.1. *In case the two locks are not identical, there exist instances for which no optimum solution is synchronised.*

We point out that if either $T_1 \leq T_2$ and $C_1 \geq C_2$, or $T_1 \geq T_2$ and $C_1 \leq C_2$, i.e. if the 'fastest' lock is also the 'largest' lock, it is easily argued that we can restrict ourselves to solutions where the number of ships in the lock with the highest capacity never exceeds the capacity of the other lock. We then obtain the setting with distinct lockage durations and equal capacity, for which an optimum synchronised solution exists, as mentioned in Section 3.5.1.

Three locks

A different generalization extends the problem setting by including a third lock. We show that there need not exist an optimum synchronised solution, even when each ship must be served by each of the three locks. Clearly, this result immediately generalizes to any setting with more than three locks.

Consider the instance described in Table 3.2 and the solution shown in Figure 3.16. It is easily verified in Figure 3.16 that the shown solution has a total waiting time of 2 time units. It follows that ships 1,2,3,6,7, and 8 must not incur any waiting time in any optimum solution since the total waiting time would be no less than 3 time units otherwise. In order to obtain a total waiting time of at most 2 time units, it then follows that ship 4 must enter a lockage (denoted by a in the figure) starting immediately upon its arrival. Then, in any synchronised solution, lock 3 starts an upwards lockage at time 9, so that ship 5 incurs a total waiting time of at least 5 time units. Thus, for this instance, no synchronised solution with a total waiting time of at most 2 time units exists.

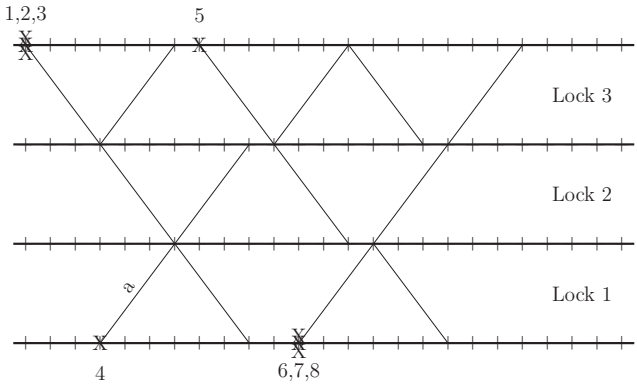


Figure 3.16: Feasible solution illustrating Observation 3.2.

Observation 3.2. *In case there are three locks, there exist instances for which no optimum solution is synchronised.*

Travel time

We now relax the assumption that the travel time in between the locks is equal to zero. Notice that the speed of ships then becomes relevant. We show that there may not exist a synchronised schedule which is optimum. Perhaps surprisingly, this result also holds when all ships travel at the same speed and thus spend the same travel time in between the locks, regardless of their direction of travel. The existence of an optimum solution which is synchronised thus highlights a noteworthy difference between the settings with and without travel time.

Consider the instance described in Table 3.3 and the feasible solution shown in Figure 3.17. It is easily verified in Figure 3.17 that, in the shown solution, ship 2 incurs a total waiting time of 1 time unit, whereas ships 1 and 3 incur no total waiting time. Also note that this solution is not synchronised, since the waiting time for ship 2 occurs in between the locks. The total waiting time in this solution equals 1 time unit. Observe that in any feasible solution with a total waiting time of at most 1 time unit, at least two of the ships incur no total waiting time. In a synchronised solution where this holds, the two ships for which this applies are thus either:

1. Ships 1 and 2. It immediately follows that the earliest time at which

Ships					
s	A_s	direction	L_s^A	L_s^D	V_s
1	0	$\mathcal{D}$	2	1	1
2	2	$\mathcal{U}$	1	2	1
3	5	$\mathcal{D}$	2	1	1

Locks			
l	T_l	C_l	S_l
1	2	∞	2
2	2	∞	-

Table 3.3: Instance for the example illustrating Observation 3.3.

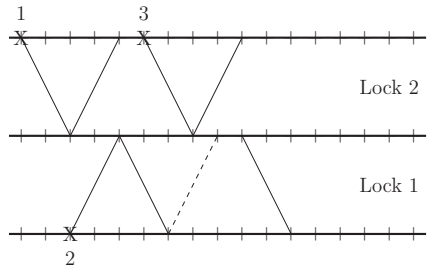


Figure 3.17: Feasible solution illustrating Observation 3.3.

lock 2 can serve ship 3 is time 8, so that the total waiting time in this solution is no less than 3 time units.

2. Ships 1 and 3. It is easily verified that, if ship 2 enters an upwards lockage in the time interval $[2, 6)$, the lockages required to serve ship 2 would overlap with one of the lockages serving ship 1 or ship 3. The earliest time at which ship 2 can enter lock 1 so that it incurs no waiting time in between the locks, is time 6. The total waiting time of such a solution is then no less than 4 time units.

3. Ships 2 and 3. It follows that lock 2 has to start a downwards lockage (serving ship 3) at time 7, and an upwards lockage (serving ship 2) at time 8: a contradiction.

We conclude that a synchronised schedule for this instance must have a total waiting time of at least 3 time units, while a feasible solution exists with a total waiting of 1 time unit.

Observation 3.3. *In case the travel time between the two locks is not equal to zero, there exist instances for which no optimum solution is synchronised.*

Not all ships pass both locks

Let us now consider the generalized setting where ships need not necessarily pass through each of the locks. We show that there does not always exist an optimum synchronised solution. In fact, this holds even in a more restrictive setting where there is one lock that must serve all ships; we refer to such a setting as a setting with a 'common lock'. The uni-directional variant of such a common lock setting is discussed in Section 3.5.3; there, it is shown that an optimum synchronised synchronised solution does exist when all ships travel in the same direction.

Consider the instance described in Table 3.4 and the feasible solution shown in Figure 3.18. Clearly, the total waiting time in this solution equals 1 time unit. Observe that in any feasible solution with a total waiting time no more than 1 time unit, at least two of the ships should incur no total waiting time. The ships for which this is the case are thus either:

1. Ships 1 and 2. It then immediately follows that lock 2 starts an upwards lockage at time 3, and that ship 3 can enter lock 2 no earlier than time 6. It follows that any solution where this is the case has a total waiting time no less than 3 time units.

2. Ships 1 and 3. A contradiction immediately follows since, in such a solution, an upwards lockage is required to start on lock 2 at time 2 as well as at time 3; clearly, this cannot be the case in any feasible solution.

3. Ships 2 and 3. It follows that lock 2 performs an upwards lockage starting at time 3. From the previous case, it is clear that ship 1 cannot be served by this lockage in any synchronised solution. The earliest time at which ship 1 can enter lock 2 in a feasible synchronised solution is thus time 7, so that such a solution has a total waiting time no less than 5 time units.

It follows from the above that any feasible synchronised solution thus has a total waiting time no less than 3 time units. Thus, there does not exist a synchronised solution which is optimum for this instance.

Observation 3.4. *In case not all ships must be served by each of the locks, there exist instances for which no optimum solution is synchronised.*

Ships				
s	A_s	direction	L_s^A	L_s^D
1	0	$\mathcal{U}$	1	2
2	0	$\mathcal{D}$	2	1
3	3	$\mathcal{U}$	2	2

Locks			
l	T_l	C_l	S_l
1	2	∞	0
2	2	∞	-

Table 3.4: Instance for the example illustrating Observation 3.4.

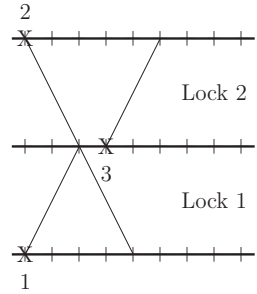


Figure 3.18: Feasible solution illustrating Observation 3.4.

3.5.3 Uni-directional traffic with a common lock

We prove that a uni-directional special case of the problem discussed in Section 3.4 can be solved in polynomial time. The problem setting discussed here considers identical locks with infinite capacity and ships travelling in a single direction at identical speeds. We assume for simplicity that ships travel at unit speed; it is easily seen that, by modifying the distance between locks, any result for unit speed extends immediately to the setting with arbitrary identical speeds. Ships may arrive at arbitrary positions along the canal. A key difference with the uni-directional variant of the setting from Section 3.4, however, is that we assume the existence of at least one lock that must serve each of the ships. We refer to this lock as the 'common lock'. That is, it is assumed that all ships travel in the upstream direction and that L_s^A and L_s^D can be arbitrarily chosen subject to the constraint that there exists at least one lock common to all ships, i.e. there is a lock $l^* \in \mathcal{L}$ satisfying $L_s^A \leq l^* \leq L_s^D$ for all $s \in \mathcal{S}$.

The underlying idea of the proposed method for solving a given instance $\mathcal{I}$ of the problem setting stated above, is as follows.

1. Construct a single-lock instance $\mathcal{I}'$ that is equivalent to instance $\mathcal{I}$.
2. Solve this single-lock instance to optimality, for example by using the procedure described in Chapter 2.
3. Extend the obtained solution to a synchronised solution for the original instance $\mathcal{I}$.

We prove that this procedure yields an optimal schedule for the original instance. In the following we describe these steps in detail and argue that the obtained solution has minimum total waiting time.

Theorem 3.4. *Problem SLS for identical locks with infinite capacity, identical ships travelling in the same direction, and a common lock, reduces to solving the uni-directional case of a single lock with infinite capacity.*

Proof. Given an instance $\mathcal{I}$, we obtain an equivalent single-lock instance $\mathcal{I}'$ as follows. Let the lockage duration of the locks in $\mathcal{I}$ be equal to T. In instance $\mathcal{I}'$, we have a single lock with lockage duration T and infinite capacity. Let the set of ships in $\mathcal{I}'$ be empty initially. For each ship s in $\mathcal{I}$, we create a ship arriving at the downstream position of the lock. The arrival time A'_s of this ship in the single-lock instance is equal to A_s minus the time needed to travel from the downstream position of the first lock in $\mathcal{I}$ to the lock where ship s arrives if no waiting time is incurred:

$$A'_s = A_s - (L_s^A - 1)T - \sum_{l=1}^{L_s^A - 1} S_l.$$

This defines instance $\mathcal{I}'$. Informally, an arrival time A_s is thus 'traced back' to the first lock by subtracting a fictitious travel time assuming that this ship s incurs no waiting time. This is visualised in Figure 3.19. Observe that the lines that trace back these arrivals do not correspond to actual lockages and thus need not be spaced without overlap. Also note that a ship may be traced back along a lock on which it need not be served in instance $\mathcal{I}$.

Next, we describe how to extend a solution for the single-lock instance $\mathcal{I}'$ to a solution for the original instance $\mathcal{I}$. In the solution for $\mathcal{I}$, we schedule lockages for the first lock at the same starting times and in the same direction as the lockages in the solution for $\mathcal{I}'$. We then extend this solution to a synchronised solution as under the assumption that each upwards lockage contains a ship travelling to the last lock. That is, let

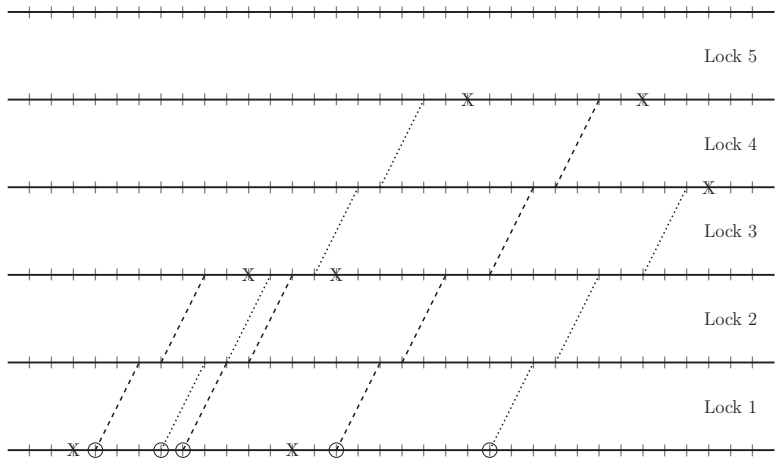


Figure 3.19: Arrival times for instance $\mathcal{I}'$. The circles mark the times of arrival in $\mathcal{I}'$ corresponding to the original arrivals at locks $2, \ldots, 5$. The travel distances equal $S_1 = 1$, $S_2 = 2$, $S_3 = 1$, and $S_4 = 2$. All ships travel to lock 5, i.e. we have $l^* = 5$.

$\mathcal{T}$ be the set of starting times of the upwards lockages of the single lock in the solution to $\mathcal{I}'$. We schedule, for each lock $l \in \mathcal{L}$ in instance $\mathcal{I}$ with $l > 1$ and for each $t \in \mathcal{T}$, an upwards lockage starting at time t', with

$$t' = t + (l-1)T + \sum_{i=1}^{l-1} S_i.$$

Additionally, we schedule downwards lockages consecutive to the added upwards lockages; i.e. at each ending time of an upwards lockage for some lock, we add a downwards lockage returning that lock to its downstream water level. An example is visualised in Figure 3.20. Observe that, by construction, the result is a synchronised schedule. Note that, since all locks have equal lockage duration, the constructed solution is feasible. Indeed, since there are no overlapping lockages in the solution to $\mathcal{I}'$, the ending times for subsequent upwards lockages are separated by at least $2T$ time units, so that no overlap is present for any of the added lockages.

In the solution to $\mathcal{I}$, we let each ship enter the first available lockage corresponding to its direction of travel. Notice that a number of lockages may be unnecessary. Indeed, since not all arrivals in $\mathcal{I}$ occur at the first

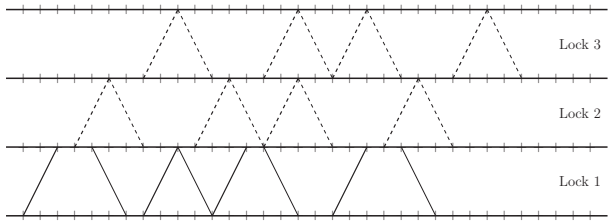


Figure 3.20: Extending a solution for the single-lock instance $\mathcal{I}'$ to a synchronised solution for instance $\mathcal{I}$ with distances $S_1 = 1$ and $S_2 = 2$.

lock, and since not all ships in $\mathcal{I}$ travel to the last lock, upwards lockages on a lock before a ship's arrival lock, or after a ship's departure lock, may in fact remain empty. Clearly, we can simply remove these empty upwards lockages and any unnecessary downwards lockages with no impact on the obtained solution.

This completes the description of the solution to $\mathcal{I}$ obtained by extending a solution to the single-lock instance $\mathcal{I}'$. Clearly, in any feasible solution to the single-lock instance, all ships are served by the lock. The constructed solution to $\mathcal{I}$ is thus feasible.

We now argue that the single-lock instance $\mathcal{I}'$ is equivalent to the original instance $\mathcal{I}$:

Lemma 3.3. *A solution with a total waiting time of at most W for instance $\mathcal{I}'$ exists if and only if a solution with a total waiting of at most W exists for the given instance $\mathcal{I}$.*

Proof. $\Rightarrow$ Consider a schedule with total waiting time W for the single-lock instance $\mathcal{I}'$, and the corresponding solution for instance $\mathcal{I}$ obtained by extending the single-lock solution as described above. For each ship in $\mathcal{I}'$, the waiting time incurred at the lock is equal to the waiting time incurred at the corresponding ship's arrival position in instance $\mathcal{I}$. Indeed, this is trivial for all ships arriving at the first lock, since both their time of arrival and the schedule of the first lock are identical in instances $\mathcal{I}$ and $\mathcal{I}'$. For all other ships, it can be seen that the difference between the arrival time of a ship s in $\mathcal{I}$ and the corresponding ship in $\mathcal{I}'$, is equal to the difference between an upwards lockage starting on the first lock and a corresponding upwards lockage of lock L_s^A.

Thus, by construction, the waiting time incurred by a ship in the solution for $\mathcal{I}'$ is equal to the waiting time of the corresponding ship in the

solution for $\mathcal{I}$. Since this holds for all ships, it follows that the constructed solution for $\mathcal{I}$ has a total waiting time of exactly W.

$\Leftarrow$ Now consider a feasible solution $\mathcal{F}$ for instance $\mathcal{I}$ with a total waiting time of at most W. Let us first argue that there exists a synchronised schedule with a total waiting time of at most W. For each ship $s \in \mathcal{S}$, let t_s be the starting time of the upwards lockage of lock l^* containing s. Next, we define A_s^* to be the latest possible time at which a ship can enter a lockage of lock L_s^A so that it reaches lock l^* at time t_s, for $s \in \mathcal{S}$. Due to the unavoidable lockage duration and the travel time between locks L_s^A and l^*, we have:

$$A_s^* = t_s - (l^* - L_s^A)T - \sum_{l=L_s^A}^{l^*-1} S_l.$$

Clearly, since solution $\mathcal{F}$ is feasible, we have $A_s \leq A_s^*$ for all ships $s \in \mathcal{S}$. Further, we define C_s^* as the earliest possible time at which a ship can leave its departure lock L_s^D if it enters lock l^* at time t_s. Due to the unavoidable lockage duration and the travel time between locks l^* and L_s^D, we have:

$$C_s^* = t_s + (L_s^D - l^* + 1)T + \sum_{l=l^*}^{L_s^D-1} S_l.$$

Let $\mathcal{T}_{l^*}$ be the set of starting times of upwards lockages of lock l^* in the given solution. We now construct a solution $\mathcal{F}^S$ as follows:

1. For all $t \in \mathcal{T}_{l^*}$, schedule an upwards lockage of lock l^* starting at time t, and schedule a downwards lockage of lock l^* starting at time $t + T$.

2. For all locks $l = l^* + 1, \ldots, l = L$, in this order, and for all $t \in \mathcal{T}_{l-1}$, schedule an upwards lockage starting at time $t + T$, and schedule a downwards lockage starting at time $t + 2T$. Define $\mathcal{T}_l$ to be the set of starting times of the upwards lockage of lock l.

3. For all locks $l = l^* - 1, \ldots, l = 1$, in this order, and for all $t \in \mathcal{T}_{l+1}$, schedule an upwards lockage starting at time $t - T$, and schedule a downwards lockage starting at time t. Define $\mathcal{T}_l$ to be the set of starting times of the upwards lockage of lock l.

Clearly, the solution $\mathcal{F}^S$ that is thus constructed is synchronised. Furthermore, since the given solution $\mathcal{F}$ is feasible and since all locks are

identical, the scheduled lockages in $\mathcal{F}^S$ do not overlap. Observe that, in $\mathcal{F}^S$, each ship $s \in \mathcal{S}$ that is served by lock l^* at time t_s, is served by its arrival lock L_s^A at time A_s^*. Since $A_s^* \geq A_s$ for all ships $s \in \mathcal{S}$, the constructed solution $\mathcal{F}^S$ is feasible. Finally observe that, in $\mathcal{F}^S$, each ship $s \in \mathcal{S}$ that is served by lock l^* at time t_s, leaves its departure lock at time C_s^*. It follows that each ship $s \in \mathcal{S}$ leaves its departure lock no later in solution $\mathcal{F}^S$ than it does in solution $\mathcal{F}$. Solution $\mathcal{F}^S$ is thus a synchronised solution with a total waiting time equal to at most W.

By the construction of instance $\mathcal{I}'$, the waiting time incurred by a ship $s \in \mathcal{S}$ in a synchronised solution for instance $\mathcal{I}$, is exactly equal to the waiting time incurred by the corresponding ship in a solution for instance $\mathcal{I}'$. There thus exists a solution with a total waiting time of at most W for $\mathcal{I}'$. □

From the lemma establishing the equivalence between an instance $\mathcal{I}$ and the corresponding instance $\mathcal{I}'$, it immediately follows that minimizing the total waiting time for instance $\mathcal{I}'$ yields an optimum solution for the original instance $\mathcal{I}$, proving the theorem. □

We note that constructing the instance $\mathcal{I}'$, as well as extending the obtained solution for the single-lock instance to a solution for the original instance, can be achieved in polynomial time. Combined with the results for a single lock described in Chapter 2, we thus obtain a polynomial time procedure for the uni-directional problem setting with identical locks, unbounded capacity, identical ships, and a common lock. Also note that the single-lock setting with infinite capacity can be solved in $O(n^2)$ time, as argued in Section 2.4. As a result, the computational complexity of the proposed algorithm does not depend on the number of locks.

3.6 Conclusion

In this chapter, we investigated the complexity of optimally scheduling a sequence of single-chamber locks with respect to the total waiting time. We showed that this problem is strongly NP-hard, even in a more restrictive setting with identical locks, non-identical ships, and uni-directional travel. Further, we also proved strong NP-hardness for a setting with identical locks, identical ships, and bi-directional travel. We reduced the gap between computationally easy and computationally hard settings by introducing a polynomial time algorithm for the setting with identical locks, identical ships, and uni-directional travel. Furthermore, we introduced the notion

of a synchronised solution, and identified two special cases for which every instance has an optimum solution which is synchronised.

We observe that the complexity of scheduling single-chamber locks in sequence is not fully characterised for all special cases that were introduced throughout this chapter. In order to close the gap between computationally easy and computationally hard problems, it may be interesting to investigate the complexity of the setting with identical locks, identical ships, and uni-directional travel. Note that this case, where ships may arrive at and depart from the canal at arbitrary positions, separates the settings for which we obtained results in Sections 3.4 and 3.5.3. Further, it may also be interesting to investigate whether the sequential lock problem with identical ships can be solved in polynomial time for a fixed number of locks, for example by means of dynamic programming.

Chapter 4

Mathematical programming for locks in sequence[1]

In this chapter, mixed integer programming models are presented for the scheduling of locks in sequence. This allows the modelling of both the waiting time and the emissions. The definition of the problem we consider is stated in Section 4.1. Section 4.2 then describes how this research relates to existing literature. Section 4.3 introduces two different mixed integer linear programming models that model our problem setting. A number of valid inequalities and other possible model improvements are discussed in Section 4.4. In order to compare the performance of scheduling a system of locks as a whole with that of a decentralized procedure which schedules the locks independently, we describe an iterative heuristic based on the solution algorithm for a single lock described in Chapter 2. A computational study, discussed in Section 4.6, compares the performance of both models and investigates the sensitivity with respect to different input parameters. In this section, we also evaluate the trade-off between the waiting time and emission objectives.

The contributions of this chapter can be summarized as follows. First, we present mixed integer programming models that model the problem

[1]The research presented in this chapter, formatted as a journal article, has been published in the European Journal of Operational Research, see Passchyn et al. (2016a) for the article version. (doi: 10.1016/j.ejor.2015.09.012)

setting with locks in sequence. Secondly, we compare the performance of the presented models with the performance of a procedure which schedules the locks independently, and thus compare the performance of a centralized and a decentralized scheduling approach. Thirdly, a computational study evaluates the performance of the models. The parameters used in the instances are derived from a real-world dataset. Note that a trade-off between reducing emissions and ships' flow times is to be expected. The computational study investigates this trade-off and analyses the potential emission reduction.

4.1 Problem definition

We follow the notation introduced in Section 1.3. Given is a set of L consecutive locks: $\mathcal{L} = \{1, \ldots, L\}$. The locks are ordered in increasing order from the downstream end of the waterway to the upstream end. We assume throughout this chapter that each lock consists of a single chamber, and that each ship travels through each of the locks. That is, upstream-travelling ships pass through the locks in the order $1, \ldots, L$, and downstream-travelling ships in the order $L, \ldots, 1$. Recall that the set of ships that arrive over time is denoted by $\mathcal{S}$, and the sets $\mathcal{U}$ and $\mathcal{D}$ refer to the upstream-travelling ships and downstream-travelling ships respectively. The arrival time of a ship s is written as A_s. All arrival times are assumed to be known. We do not assume an initial position for the locks, i.e. the initial water level of each lock may equal the lock's upstream water level as well as the lock's downstream water level, although the models presented in Section 4.3 can be easily extended to enforce a fixed initial position.

Also recall that S_l denotes the length of the waterway section between locks l and $l+1$, and that each ship $s \in \mathcal{S}$ has a fixed minimum and maximum travel speed, represented by $V_s^{\min}$ and $V_s^{\max}$ respectively. Ship s may thus travel at an arbitrary speed contained in the interval $[V_s^{\min}, V_s^{\max}]$. While the ships can travel at a different speed on different sections of the canal, we assume that each ship maintains a constant speed within each section. Additionally, when emissions are considered, each ship $s \in \mathcal{S}$ may also have an imposed deadline D_s, before which it must have left the last lock that it needs to pass.

The problems we consider are to find an optimum feasible solution with respect to the objective functions discussed in Sections 4.1.1 and 4.1.2.

4.1.1 Minimizing total flow time

As in the previous chapters, one objective could be to minimize the total flow time, i.e. the elapsed time between the ship's arrival at the first lock and departure from the last lock in its direction of travel, summed over all ships. Given this objective function we can easily see that we can fix the speed of each ship s to its maximum value $V_s^{\max}$ without sacrificing optimality.

4.1.2 Minimizing emissions

An alternative objective is to minimize the total emissions or the total fuel cost. While the emission of greenhouse gases may be an optimization criterion for governments or waterway organizations, the strongly related fuel consumption is also an important economical factor for ship operators. Since both the fuel consumption and emissions per kilometre travelled increase at higher ship speeds, it is important to take the ship's speed into account when considering this objective. It is frequently assumed that the emissions are directly proportional to the fuel consumption, so that minimizing the emissions is equivalent to minimizing the fuel cost; we thus ignore the fuel cost in the remainder of this chapter. We refer to Section 4.2 for literature related to variable ship speeds and the related emissions. We assume that an emission function $E(v)$ is given which expresses the emission of pollutants, in tonnes per km, as a function of the ship speed v for $v > 0$. Note that this function E depends on many external factors and is generally non-linear due to the ship's increasing resistance in the water. In idealized settings, the emission per km is typically a convex function of ship speed. For the emissions expressed per time unit as a function of ship speed, a cubic relationship bounded by a constant minimum at low speeds is often assumed, e.g. by Fagerholt et al. (2010). When this relationship is expressed in emissions per unit of distance travelled as a function of ship speed, this corresponds to a quadratic relationship, bounded by a hyperbola for low speed values. A qualitative illustration can be seen in Figure 4.1. Clearly, it is suboptimal to sail at a speed below the threshold value where the minimum is attained, since increasing both the ship speed and subsequent idle time then yields a better solution. Regardless of the existence of such a threshold speed, imposing a minimum value for the speed is also justified in order to guarantee sufficient manoeuvrability.

Let $v_{s,p}$ denote the speed of ship s along segment p. The total emission

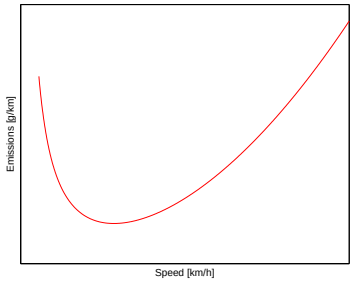


Figure 4.1: Qualitative graph of an emission function exhibiting a minimum at a threshold speed. Note that emissions are expressed per kilometre travelled.

of greenhouse gases E_{tot} can then be computed as follows:

$$E_{\text{tot}} = \sum_{s \in \mathcal{S}} \sum_{l \in \mathcal{L} \setminus \{L\}} S_l E(v_{s,l}).$$

A similar approach can be applied in order to minimize fuel consumption, which may be more desirable from the ship operator's point of view.

We note that assuming constant speed on the individual sections cannot be justified for arbitrary functions $E(v)$, although it is easily seen that a constant speed is optimal for any convex emission function, see Hvattum et al. (2013).

4.2 Relation to literature

Literature on the scheduling of a sequence of locks remains scarce. Petersen and Taylor (1988) extend a dynamic programming approach for a single lock to a constructive heuristic for a set of locks in sequence, although batching is ignored, i.e. each lock is assumed to have unit capacity. Only recently, a formal definition for the problem of scheduling locks arranged in a sequence was described by Prandtstetter et al. (2015), accompanied by a variable neighbourhood search approach to obtain heuristic solutions. Despite this apparent gap in literature, however, the potential of scheduling a series of locks as a whole, as opposed to each lock individually, is recognized by different stakeholders. The Dutch waterway management organization

Rijkswaterstaat, for example, is shifting its focus from decentralized lock operations towards the fluent operation of certain 'corridors' as a whole, see Kunst (2013).

In recent years, the operational speed of intercontinental container ships has been decreased to improve fuel efficiency, a practice referred to as 'slow steaming'. A large body of literature exists on optimizing ship speed for maritime shipping, see e.g. Ronen (1982) for some early results on this topic, and Psaraftis and Kontovas (2013) for a recent survey. In the context of ocean liner shipping, a reduction in ship speeds clearly translates to an increase in the total travel time. On inland waterways however, ships are likely to incur waiting time near bottlenecks such as locks. This provides ships with the opportunity of decreasing their maximal speed on each of the sections along the canal while their total time spent inside the canal, i.e. the flow time, remains unchanged. A paper by Ting and Schonfeld (1999) mentions the strategy of reducing ship speed to avoid idle time when a single lock is present; significant economic benefits are reported. Our results in this chapter extend this by showing how to integrate the decisions for multiple locks. To the best of our knowledge, the models presented in this chapter are the first MIP models for lock scheduling which include the ship speed as a variable and allow to take the emissions and fuel cost into account. In the case of minimizing emissions, communicating the outcome of the models to the ships allows lowering the ship speed when locks are known to be unavailable; this avoids unnecessary idle time where ships would arrive at a lock before it is available. At the same time, this yields a reduction in fuel cost as well as pollutant emissions.

4.3 Mathematical programming models

We introduce here two distinct MIP formulations representing the problems introduced in Section 4.1. In this section, we present a basic formulation for both models. Valid inequalities and other model improvements are presented in Section 4.4.

To state the travel time, which appears in the constraints of both models, as a linear expression of the variables, we introduce the variables $\bar{v}_{s,l}$, which equals the reciprocal of $v_{s,l}$, i.e. $\bar{v}_{s,l} = 1/v_{s,l}$. The travel time for ship s along the section between locks l and $l+1$ then equals $\bar{v}_{s,l} S_l$. To characterize the emissions, we use the function $\bar{E}(\bar{v})$, which expresses the emissions as a function of the reciprocal of ship speed. To obtain a linear model, we use a piecewise linear approximation of $\bar{E}(\bar{v})$. Note that, in

general, using a piecewise linear approximation may introduce additional variables as the number of segments increases. If the piecewise function is convex and is to be minimized, however, it suffices to introduce a single additional variable and a set of additional constraints. In order to reduce the notational burden we do not give the linearisation here and use $\bar{E}(\bar{v})$ when we refer to the approximation. For an example and additional details on how to compute $E(v)$ and $\bar{E}(\bar{v})$, we refer to Section 4.6.3.

Note that even with S_p and $v_{s,p}$ integral, the travel time for some sections may be fractional. From a practical point of view, however, it might not make sense to schedule lock movements with a higher precision than the unit in which the arrival times are expressed, e.g. minutes. The time-indexed model described below allows lockages to start only at integral moments in time, whereas the second model allows arbitrary starting times for the lock movements.

4.3.1 Time-indexed formulation

For the model formulation we consider a discretised planning horizon. We consider a set $\mathcal{T} = \{0, \ldots, T\}$ of points of time. In the model then we restrict ourselves to start times from $\mathcal{T}$ for each lock movement.

We obtain a bound T on the latest possible lockage start time as the sum of the latest arrival time, maximum total travel time, total lockage time, and the maximum waiting time. An upper bound for the latter is derived as the maximum waiting time that a ship can spend at an artificial lock having capacity $C_{\min}$ and lockage duration $P_{\max}$ and assuming that all other ships travelling in the same direction are waiting when s arrives.

Let W_u and W_d denote, respectively, the maximum waiting time for an upstream-travelling ship and for a downstream-travelling ship, and let $C_{\min} = \min_{l \in \mathcal{L}} C_l$ and $P_{\max} = \max_{l \in \mathcal{L}} P_l$. Then

$$W_u = \left\lfloor \frac{|\mathcal{U}|}{C_{\min}} \right\rfloor 4P_{\max},$$
$$W_d = \left\lfloor \frac{|\mathcal{D}|}{C_{\min}} \right\rfloor 4P_{\max},$$

and with $\mathrm{TR}_{\max}$ equal to the maximum time needed to pass through the

entire canal without waiting, we obtain

$$T = \max\Bigg(\max_{s\in\mathcal{U}} A_s + 2 * P_{\text{total}} + \text{TR}_{\max} + W_u,$$
$$\max_{s\in\mathcal{D}} A_s + 2 * P_{\text{total}} + \text{TR}_{\max} + W_d\Bigg).$$

For each ship s and each lock l we can further reduce the relevant points in time to the interval between earliest possible arrival time and latest feasible departure time of s at l. We thus obtain a set $\mathcal{T}_{s,l}$ for each ship s and lock l.

In addition to the variables $\bar{v}_{s,l}$ introduced in Section 4.1.2, we define the following binary decision variables: For each $s \in \mathcal{S}$, $l \in \mathcal{L}$, $t \in \mathcal{T}_s$, let

$$x_{s,l,t} = \begin{cases} 1 & \text{if at time } t, \text{ lock } l \text{ starts a lockage serving ship } s, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the completion time c_s can then be expressed as follows:

$$c_s = \begin{cases} \sum_{t\in\mathcal{T}_{s,L}} x_{s,L,t}(t + P_L) & \text{if } s \in \mathcal{D}, \\ \sum_{t\in\mathcal{T}_{s,1}} x_{s,1,t}(t + P_1) & \text{if } s \in \mathcal{U}. \end{cases}$$

Model 1 shows the mathematical formulation of the time-indexed model. For a schedule to be feasible, each ship should pass each of the locks, as imposed by (4.2). Notice that this constraint also implicitly ensures that the arrival time and deadline for each ship are respected, since only variables in $\mathcal{T}_{s,l}$ are considered. Further, all locks must be passed in the correct order, i.e. a ship must arrive at a lock before it can enter that lock. We achieve this by imposing constraints (4.3) and (4.4). In these inequalities, the left hand side reflects the time difference between the starting time of the lock movement that handles ship s at lock l, and the ending time of the lockage that handles ship s at the preceding lock.

Additionally, a lockage can only start when the lock is in the appropriate position and not currently moving, i.e. lockages of the same lock should not overlap in time. We impose this by adding the constraints (4.5) to (4.7). Figures 4.2 and 4.3 give a visual representation of these constraints. In the figures, for any pair of ships s_1, s_2 travelling in the appropriate direction, a movement for ship s_2 cannot start in the indicated interval if the x variable corresponding to s_1 equals one. Note that constraints (4.6) and (4.7), which concern ships travelling in the same direction, allow multiple ships to be handled at the same time.

$$\text{Minimize} \sum_{s\in\mathcal{S}} \Big(c_s - A_s\Big) \quad \text{or} \quad \text{Minimize} \sum_{s\in\mathcal{S}} \sum_{l\in\mathcal{L}\setminus\{L\}} S_l \bar{E}(\bar{v}_{s,l}) \tag{4.1}$$

Subject to:

$$\sum_{t\in\mathcal{T}_{s,l}} x_{s,l,t} = 1 \qquad \forall s \in \mathcal{S}, l \in \mathcal{L} \tag{4.2}$$

$$\sum_{t\in\mathcal{T}_{s,l}} (t\, x_{s,l,t}) - \sum_{t\in\mathcal{T}_{s,l+1}} (t\, x_{s,l+1,t}) \geq P_{l+1} + \bar{v}_{s,l} S_l \qquad \forall s \in U, \forall l \in \mathcal{L}\setminus\{L\} \tag{4.3}$$

$$\sum_{t\in\mathcal{T}_{s,l}} (t\, x_{s,l,t}) - \sum_{t\in\mathcal{T}_{s,l-1}} (t\, x_{s,l-1,t}) \geq P_{l-1} + \bar{v}_{s,l-1} S_{l-1} \qquad \forall s \in \mathcal{D}, \forall l \in \mathcal{L}\setminus\{1\} \tag{4.4}$$

$$x_{s_1,l,t} + \sum_{\substack{\tau\in\mathcal{T}_{s_2,l}\,\cap \\ \{t-P_l+1,\ldots,t+P_l-1\}}} x_{s_2,l,\tau} \leq 1 \qquad \forall l \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{D}, t \in \mathcal{T}_{s_1,l} \tag{4.5}$$

$$x_{s_1,l,t} + \sum_{\substack{\tau\in\mathcal{T}_{s_2,l}\,\cap \\ (\{t-2P_l+1,\ldots,t-1\}\cup \\ \{t+1,\ldots,t+2P_l-1\})}} x_{s_2,l,\tau} \leq 1 \qquad \forall l \in \mathcal{L}, s_1, s_2 \in \mathcal{U}, t \in \mathcal{T}_{s_1,l} \tag{4.6}$$

$$x_{s_1,l,t} + \sum_{\substack{\tau\in\mathcal{T}_{s_2,l}\,\cap \\ (\{t-2P_l+1,\ldots,t-1\}\cup \\ \{t+1,\ldots,t+2P_l-1\})}} x_{s_2,l,\tau} \leq 1 \qquad \forall l \in \mathcal{L}, s_1, s_2 \in \mathcal{D}, t \in \mathcal{T}_{s_1,l} \tag{4.7}$$

$$\sum_{s\in S, t\in\mathcal{T}_{s,l}} x_{s,l,t} \leq C_l \qquad \forall l \in \mathcal{L}, t \in \bigcup_{s\in\mathcal{S}} \mathcal{T}_{s,l} \tag{4.8}$$

$$\frac{1}{V_s^{\max}} \leq \bar{v}_{s,l} \leq \frac{1}{V_s^{\min}} \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}\setminus\{L\} \tag{4.9}$$

$$x_{s,l,t} \in \{0,1\} \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}, t \in \mathcal{T}_{s,l} \tag{4.10}$$

$$\bar{v}_{s,l} \in \mathbb{R}^+ \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}\setminus\{L\} \tag{4.11}$$

Model 1: Time-indexed model

Finally, the locks' capacity restrictions, the limits on attainable ship speed, and the domain of all variables are trivially modelled by constraints (4.8) to (4.10).

We point out that this model contains $O(SLT)$ binary variables, $O(SL)$ real variables, and $O(S^2LT)$ constraints. We refer to Section 4.4.1 for a number of model improvements that enhance the performance of the time-indexed model (4.1) to (4.11).

4.3.2 Lockage-based formulation

A notable disadvantage of the time-indexed model is that the number of (binary) variables grows as the time horizon increases. For a small discretisation step or when arrival times are large, the value for the upper

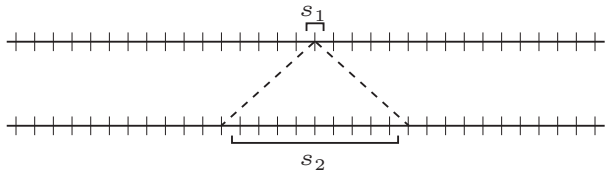


Figure 4.2: Visualisation of constraints (4.5), ships travelling in opposite direction.

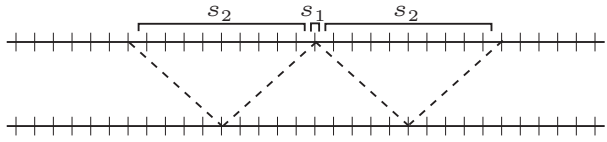


Figure 4.3: Visualisation of constraints (4.6) and (4.7), ships travelling in the same direction.

bound T and thus the number of variables and the computation time to find an optimal solution, may grow prohibitively large. We introduce an alternative formulation that does not use a time index for the variables, and instead numbers the possible lockages. It is clear that for each lock, the number of lockages in an optimal solution does not need to be greater than $2S$. Furthermore, we may also use the bound T introduced with the TI model to bound the number of lockages:

$$K = \min\left(2S, \left\lceil \frac{T + \max_{l \in \mathcal{L}} P_l}{\min_{l \in \mathcal{L}} P_l} \right\rceil\right).$$

We thus define the set $\mathcal{K} = \{1, \ldots, K\}$ to identify the lockages of each lock. In fact, it is easy to see that we can replace $\mathcal{K}$ with an individual expression $\mathcal{K}_l$ for each lock, but we ignore this for simplicity of notation.

In addition to the variables $\bar{v}_{s,p}$, we introduce the following decision variables for each $s \in \mathcal{S}, l \in \mathcal{L}, k \in \mathcal{K}$:

$$z_{s,l,k} = \begin{cases} 1 & \text{if ship } s \text{ is handled by the } k\text{'th lockage of lock } l, \\ 0 & \text{otherwise.} \end{cases}$$

$t_{l,k}$ equals the starting time of the k'th lockage of lock l,

c_s equals the completion time of ship s.

Note that the number of variables does not increase with T. Model 2 presents the complete formulation. For ease of notation, we do not explicitly mention that $\kappa, \kappa_1, \kappa_2 \in \mathcal{K}$ whenever they are used.

$$\text{Minimize} \sum_{s\in\mathcal{S}} \Big(c_s - A_s\Big) \quad \text{or} \quad \text{Minimize} \sum_{s\in\mathcal{S}} \sum_{l\in\mathcal{L}\setminus\{L\}} S_l \bar{E}(\bar{v}_{s,l}) \tag{4.12}$$

Subject to:

$$D_s \geq c_s \geq t_{1,k} + P_1 - M_s^u(1 - \sum_{\kappa\geq k} z_{s,1,\kappa}) \qquad \forall s \in \mathcal{U}, k \in \mathcal{K} \tag{4.13}$$

$$D_s \geq c_s \geq t_{L,k} + P_L - M_s^d(1 - \sum_{\kappa\geq k} z_{s,L,\kappa}) \qquad \forall s \in \mathcal{D}, k \in \mathcal{K} \tag{4.14}$$

$$\sum_{k\in\mathcal{K}} z_{s,l,k} = 1 \qquad \forall s \in \mathcal{S}, l \in \mathcal{L} \tag{4.15}$$

$$t_{l,k} \geq t_{l,k-1} + P_l \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}\setminus\{1\} \tag{4.16}$$

$$z_{s_1,l,k} + z_{s_2,l,k} \leq 1 \qquad \forall s_1 \in \mathcal{U}, s_2 \in \mathcal{D} l \in \mathcal{L}, k \in \mathcal{K} \tag{4.17}$$

$$z_{s_1,l,k-1} + z_{s_2,l,k} \leq 1 \qquad \forall s_1, s_2 \in \mathcal{U}, l \in \mathcal{L}, k \in \mathcal{K}\setminus\{1\} \tag{4.18}$$

$$z_{s_1,l,k-1} + z_{s_2,l,k} \leq 1 \qquad \forall s_1, s_2 \in \mathcal{D}, l \in \mathcal{L}, k \in \mathcal{K}\setminus\{1\} \tag{4.19}$$

$$t_{L,k} \geq A_s \sum_{\kappa\leq k} z_{s,L,\kappa} \qquad \forall s \in \mathcal{U}, k \in \mathcal{K} \tag{4.20}$$

$$t_{1,k} \geq A_s \sum_{\kappa\leq k} z_{s,1,\kappa} \qquad \forall s \in \mathcal{D}, k \in \mathcal{K} \tag{4.21}$$

$$t_{l,k_1} \geq t_{l+1,k_2} + P_{l+1} + \bar{v}_{s,l} S_l - M_{k_1,k_2}(2 - \sum_{\kappa\leq k_1} z_{s,l,\kappa} - \sum_{\kappa\geq k_2} z_{s,l+1,\kappa}) \qquad \forall s \in \mathcal{U}, l \in \mathcal{L}\setminus\{L\}, k_1, k_2 \in \mathcal{K} \tag{4.22}$$

$$t_{l,k_1} \geq t_{l-1,k_2} + P_{l-1} + \bar{v}_{s,l-1} S_{l-1} - M_{k_1,k_2}(2 - \sum_{\kappa\leq k_1} z_{s,l,\kappa} - \sum_{\kappa\geq k_2} z_{s,l-1,\kappa}) \qquad \forall s \in \mathcal{D}, l \in \mathcal{L}\setminus\{1\}, k_1, k_2 \in \mathcal{K} \tag{4.23}$$

$$\sum_{s\in S} z_{s,l,k} \leq C_l \qquad \forall l \in \mathcal{L}, k \in \mathcal{K} \tag{4.24}$$

$$\frac{1}{V_s^{\max}} \leq \bar{v}_{s,l} \leq \frac{1}{V_s^{\min}} \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}\setminus\{L\} \tag{4.25}$$

$$z_{s,l,k} \in \{0,1\} \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}, k \in \mathcal{K} \tag{4.26}$$

$$t_{l,k} \in \mathbb{R}_+ \qquad \forall l \in \mathcal{L}, k \in \mathcal{K} \tag{4.27}$$

$$c_s \in \mathbb{R}_+ \qquad \forall s \in \mathcal{S} \tag{4.28}$$

Model 2: Lockage-based model

Notice that the model introduces M_s^u, M_s^d, and M_{k_1,k_2} as big-M values. To define their value, we first define the parameter $\overline{T}$, which gives an upper bound on the latest starting time of any lock movement. Note that this value may exceed the T introduced for the time-indexed model. We have

at least $\lceil \frac{S}{C_{\min}} \rceil$ non-empty lockages, thus:

$$\overline{T} = T + P_{\max} + \left(K - \left\lceil \frac{S}{C_{\min}} \right\rceil \right) P_{\max}$$

We now discuss the individual constraints. Inequalities (4.13) and (4.14) ensure that the completion time of each ship, used in the objective function, is consistent with the timing of the last lockage the ship passes through and enforce the deadlines for each ship. The values M_s^u and M_s^d introduced here give an upper bound on the difference between the latest possible starting time for a lockage and the earliest completion time for a ship s which is upstream-travelling or downstream-travelling respectively. With $\mathrm{TR}_{\min}$ equal to the minimum time required to cross the entire canal, we can write:

$$M_s^u = \overline{T} + P_1 - A_s - \mathrm{TR}_{\min}$$
$$M_s^d = \overline{T} + P_L - A_s - \mathrm{TR}_{\min}$$

Constraint (4.15) guarantees that each ship passes through each of the locks and constraint (4.16) ensures that lockages do not overlap in time.

The next set of inequalities, (4.17) to (4.19), ensures that a lockage can only handle ships in a single direction with each lockage, and that no two consecutive lockages carry ships in the same direction. Note that because the only way to characterize the direction of a lockage is to consider the direction of ships inside the lockage, the direction of lockages need not necessarily alternate when empty lockages are present. Using these constraints, however, the model does guarantee that all non-empty lockages satisfy all requirements for a feasible solution.

Obviously, the locks should be passed in the correct order. This is imposed by specifying that the waiting time each ship incurs at each position must be non-negative. Constraints (4.20) and (4.21) ensure this for the outer positions where the arrival times are known, whereas constraints (4.22) and (4.23) do the same for the middle positions. The big-M value M_{k_1,k_2} gives an upper bound on the difference in starting time between the lockages k_1 and k_2:

$$M_{k_1,k_2} = \overline{T} - (k_1 - 1)P_{\min} - (K - k_2 - 1)P_{\min}$$

Finally, the capacity constraint (4.24) and the domain restrictions for the variables are again straightforward to specify.

The lockage-based model involves $O(S^2L)$ binary variables, $O(SL)$ real variables, and $O(S^3L)$ constraints. We refer to Section 4.4.2 for a number of performance improvements for the lockage-based model (4.12) to (4.28).

4.3.3 First-come first-served constraints

Notice that, in general, both models allow ships to overtake each other. That is, for two ships $s_1, s_2 \in \mathcal{S}$ with $A_{s_1} < A_{s_2}$, the model allows ship s_2 to enter any of the locks strictly before ship s_1 enters that lock. In practice, a first-come first-served rule is often enforced for ships travelling in the same direction, see e.g. Smith et al. (2009) and Ting and Schonfeld (2001). Furthermore, if there is no difference between the ships other than their time of arrival, it is easily argued that at least one solution with minimal flow time exists that satisfies first-come first-served. Since this is the case in the problem we consider here, we order $\mathcal{U}$ and $\mathcal{D}$ by arrival time and add the following constraints to the TI-model:

$$\sum_{\tau \leq t} x_{s_1,l,\tau} \geq \sum_{\tau \leq t} x_{s_2,l,\tau} \quad \forall s_1 < s_2 \in \mathcal{U}, l \in \mathcal{L}, t \in \mathcal{T} \tag{4.29}$$

$$\sum_{\tau \leq t} x_{s_1,l,\tau} \geq \sum_{\tau \leq t} x_{s_2,l,\tau} \quad \forall s_1 < s_2 \in \mathcal{D}, l \in \mathcal{L}, t \in \mathcal{T} \tag{4.30}$$

For the LB-model, we add the following:

$$\sum_{\kappa \leq k} z_{s_1,l,\kappa} \geq \sum_{\kappa \leq k} x_{s_2,l,\kappa} \quad \forall s_1 < s_2 \in \mathcal{U}, l \in \mathcal{L}, \kappa \in \mathcal{K} \tag{4.31}$$

$$\sum_{\kappa \leq k} z_{s_1,l,\kappa} \geq \sum_{\kappa \leq k} x_{s_2,l,\kappa} \quad \forall s_1 < s_2 \in \mathcal{D}, l \in \mathcal{L}, \kappa \in \mathcal{K} \tag{4.32}$$

We note that this FCFS assumption need not hold in general. For example, in the extension discussed above where ship-dependent handling times are present, it may be optimal for ships with a small handling time to overtake ships with a larger handling time. It is also worth noting that some of the constraints introduced in Models 1 and 2 can be tightened by implicitly taking this FCFS property into account. In order to preserve the flexibility of the original model when FCFS does not hold, however, we do not modify the original constraints.

4.3.4 Model extensions

We note that the models presented above do not include all practical considerations which may occur in a specific real-world setting. The

generality of both models, however, allows such extensions to be included with relative ease, although the required computational effort is likely to increase significantly as a result. We briefly mention some of these additional issues that may arise:

- ships may arrive or leave the system at arbitrary position, as opposed to the first and last lock only.
- the lockage time may depend on the number of ships inside the lock. In particular, empty lockages may be performed very efficiently.
- the water velocity may affect the lockage time depending on whether the water level is raised or lowered,
- the water velocity may also effect the emission characteristics of ships depending on their direction of travel.

4.4 Model improvements

4.4.1 Improvements for the time-indexed model

We present here a number of enhancements to the time-indexed model. A first performance improvement can be gained by reducing the number of variables. Next, we discuss a number of valid inequalities which could be added to obtain a stronger LP-relaxation.

Reducing the number of variables

As argued before we can assume the ship speeds to be fixed to the maximum speed when we consider the objective to minimize total flow time. It can be easily argued that there exists an optimal solution where each lockage either starts at the arrival time of some ship that is in the lockage, or immediately follows the lockage that precedes it. We can make use of this fact in order to obtain $\mathcal{T}'_{s,l}$ by eliminating all elements from $\mathcal{T}_{s,l}$ where no ship can arrive and where no lockage can end. Generating all remaining variables can be done by starting from the arrival times of ships as the initial possible lockage start times. Adding all multiples of the lockage time and the appropriate travel time to each of these starting times yields more possible lockage start times for the neighbouring locks. By repeating this while always considering the earliest possible lockage start time first, we obtain a set $\mathcal{T}' \subseteq \mathcal{T}$. Since we can also compute the earliest possible

time $e_{s,l}$ for a ship s to arrive at the position where it enters lock l, and the latest possible time $l_{s,l}$ for ship s to enter lock l in time to reach the deadline D_s, we obtain the set $\mathcal{T}'_s = \{t \mid t \in \mathcal{T}', e_{s,l} \leq t \leq l_{s,l}\}$ containing all relevant time units at which ship s may move. Employing $\mathcal{T}'_{s,l}$ instead of $\mathcal{T}_{s,l}$ avoids including many unnecessary variables and yields a drastic performance improvement. We refer to Section 4.6 for a comparison of the TI model with and without the application of this procedure.

Valid inequalities

In the time-indexed model, the constraints that avoid overlap of the lockages contain a single variable $x_{s_1,l,t}$ for ship s_1, and a sum over a time interval for ship s_2. A generalization of these constraints can be obtained by summing over an interval for both ships s_1 and s_2:

$$\sum_{\substack{\tau \in \mathcal{T}_{s_1,l} \cap \\ \{t,\ldots,t+p\}}} x_{s_1,l,\tau} + \sum_{\substack{\tau \in \mathcal{T}_{s_2,l} \cap \\ \{t-P_l+p+1,\ldots,t+P_l-1\}}} x_{s_2,l,\tau} \leq 1 \tag{4.33}$$
$$\forall l \in \mathcal{L}, s_1 \in \mathcal{D}, s_2 \in \mathcal{U}, t \in \mathcal{T}_{s_1,l}, p \in \{0,\ldots,2P_l-2\}$$

$$\sum_{\substack{\tau \in \mathcal{T}_{s_1,l} \cap \\ \{t,\ldots,t+p\}}} x_{s_1,l,\tau} + \sum_{\substack{\tau \in \mathcal{T}_{s_2,l} \cap \\ \{t-P_l+p+1,\ldots,t+P_l-1\}}} x_{s_2,l,\tau} \leq 1 \tag{4.34}$$
$$\forall l \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{D}, t \in \mathcal{T}_{s_1,l}, p \in \{0,\ldots,2P_l-2\}$$

It is not hard to see that we do not exclude any feasible solution, while some solutions from the LP-relaxation of the original formulation are excluded. We can use a similar approach when s_1 and s_2 are travelling in the same direction:

$$\sum_{\substack{\tau \in \mathcal{T}_{s_1,l} \cap \\ \{t,\ldots,t+p\}}} x_{s_1,l,\tau} + \sum_{\substack{\tau \in \mathcal{T}_{s_2,l} \cap \\ (\{t-2P_l+p+1,\ldots,t-1\} \cup \\ \{t+p+1,\ldots,t+2P_l-1\})}} x_{s_2,l,\tau} \leq 1 \tag{4.35}$$
$$\forall l \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{U}, t \in \mathcal{T}_{s_1,l}, p \in \{0,\ldots,2P_l-2\}$$

$$\sum_{\substack{\tau \in \mathcal{T}_{s_1,l} \cap \\ \{t,\ldots,t+p\}}} x_{s_1,l,\tau} + \sum_{\substack{\tau \in \mathcal{T}_{s_2,l} \cap \\ (\{t-2P_l+p+1,\ldots,t-1\} \cup \\ \{t+p+1,\ldots,t+2P_l-1\})}} x_{s_2,l,\tau} \leq 1 \tag{4.36}$$
$$\forall l \in \mathcal{L}, s_1 \in \mathcal{D}, s_2 \in \mathcal{D}, t \in \mathcal{T}_{s_1,l}, p \in \{0,\ldots,2P_l-2\}.$$

Graphically, we can represent these constraints as shown in Figures 4.4 and 4.5.

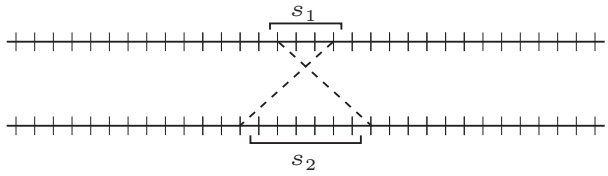


Figure 4.4: Visualisation of constraints (4.33) and (4.34), ships travelling in opposite direction.

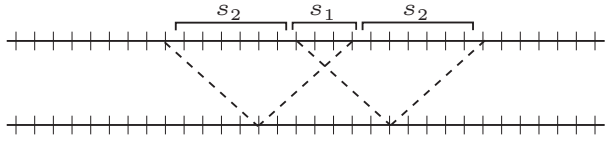


Figure 4.5: Visualisation of constraints (4.35) and (4.36), ships travelling in the same direction.

Note that for $p = 0$, constraints (4.33) to (4.36) imply constraints (4.5) to (4.7) of the original IP model. Although constraint (4.33) suffices to prevent overlap of alternating lock movements, (4.34) repeats this constraint for ships moving in the opposite direction to cut away additional LP-solutions. A tighter LP relaxation can thus be obtained by replacing constraints (4.5) to (4.7) by constraints (4.33) to (4.36).

Experiments with this stronger formulation show that, even though the LP-relaxation yields a stronger bound, the solution time does not decrease on average, which can be explained by the computational burden of generating and processing these $O(P_{\max} S^2 LT)$ additional constraints. However, these stronger constraints may still be of interest when the focus is on finding a good LP solution, or if a separation procedure could be applied in order to avoid generating many of the non-binding constraints.

4.4.2 Improvements for the lockage-based model

We introduce here a reformulation of the LB-model that allows to fix some of the variables, as well as different valid inequalities that strengthen the LP-bound.

Because lockages must alternate direction, we can impose that even-numbered lockages contain only ships in $\mathcal{U}$ and all odd-numbered lockages contain only ships in $\mathcal{D}$, or vice versa. We can do this by enforcing all

ships in $\mathcal{U}$ and $\mathcal{D}$ to be restricted to even and odd numbered lockages respectively, while allowing for an additional empty lockage that ends at time $t = 0$. This additional lockage is required to allow the first lockage starting not earlier than $t = 0$ to be an upstream as well as a downstream lockage. We impose that:

$$z_{s,l,k} = 0 \qquad \forall s \in \mathcal{U}, l \in \mathcal{L}, k \in \mathcal{K} \cup \{K+1\} : \ k \text{ is even} \tag{4.37}$$

$$z_{s,l,k} = 0 \qquad \forall s \in \mathcal{D}, l \in \mathcal{L}, k \in \mathcal{K} \cup \{K+1\} : \ k \text{ is odd}, \tag{4.38}$$

which fixes approximately half of the original $z_{s,l,k}$ variables to zero, effectively removing them from the model, at the cost of increasing K by one and some adjustments to inequalities (4.20) and (4.21) discussed below. Furthermore, we may remove constraints (4.17)-(4.19) from the formulation, since they are no longer relevant.

Some care must be taken to allow the first lockages at locks 1 and L to start at time $-P_1$ and $-P_L$. Constraints (4.20) and (4.21) imply non-negative starting times for all lockages. We can substitute the following for these two constraints:

$$t_{1,k} \geq \sum_{\kappa \leq k} z_{s,1,\kappa}(A_s + P_1) - P_1 \qquad \forall s \in \mathcal{D}, k \in \mathcal{K} \tag{4.39}$$

$$t_{L,k} \geq \sum_{\kappa \leq k} z_{s,L,\kappa}(A_s + P_L) - P_L \qquad \forall s \in \mathcal{U}, k \in \mathcal{K}. \tag{4.40}$$

In addition, the big-M value M_{k_1,k_2} should be increased by $P_{\max}$.

In the remainder, we will discuss a number of valid inequalities to enhance this reformulated LB-model. To express these valid inequalities, we introduce two new sets of continuous variables:

$a_{s,l} \in \mathbb{R}$ equals the earliest time at which ship s can enter lock l,

$b_{s,l,\kappa} \in \mathbb{R}$ equals $\begin{cases} 0 & \text{if } a_s \leq t_{l,\kappa}, \\ a_{s,l} - t_{l,\kappa} & \text{if } a_s > t_{l,\kappa}. \end{cases}$

We have the following constraints linking $a_{s,l}$ and $b_{s,l,\kappa}$ to the original variables:

$$a_{s,1} = A_s \qquad \forall s \in \mathcal{D} \tag{4.41}$$

$$a_{s,L} = A_s \qquad \forall s \in \mathcal{U} \tag{4.42}$$

$$a_{s,l} \geq t_{l-1,k} + P_{l-1} + S_{l-1}\bar{v}_{s,l-1} - (\overline{T} - E_{s,l}) \sum_{\kappa<k} z_{s,l-1,\kappa} \qquad \forall s \in \mathcal{D}, l \in \mathcal{L} \setminus \{1\}, k \in \mathcal{K} \tag{4.43}$$

$$a_{s,l} \geq t_{l+1,k} + P_{l+1} + S_l\bar{v}_{s,l} - (\overline{T} - E_{s,l}) \sum_{\kappa<k} z_{s,l+1,\kappa} \qquad \forall s \in \mathcal{U}, l \in \mathcal{L} \setminus \{L\}, k \in \mathcal{K} \tag{4.44}$$

$$b_{s,l,k} \geq a_{s,l} - t_{l,k} \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}, k \in \mathcal{K} \tag{4.45}$$

$$b_{s,l,k} \geq 0 \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}, k \in \mathcal{K} \tag{4.46}$$

where $E_{s,l}$ equals the earliest time at which ships s can arrive at lock l.

Using the $a_{s,l}$ variables, constraints (4.22) and (4.23) in the original LB-model can be replaced by the following:

$$t_{l,k} \geq a_{s,l} - M_{k,l}^{LB+} \sum_{\kappa>k} z_{s,l,\kappa} \quad \forall s \in \mathcal{S}, l \in \mathcal{L}, k \in \mathcal{K} \tag{4.47}$$

with $M_{k,l}^{LB+} = \overline{T} + P_l - (k-1)P_l$. Notice that the value $M_{k,l}^{LB+}$ is smaller than the value of M_{k_1,k_2} in the original LB-model.

We now express a number of valid inequalities which further strengthen the formulation of the LB-model. While the model allows for any number of empty lockages provided that they do not overlap, it is trivial to argue that two consecutive empty movements can be removed from any schedule without affecting the solution value. We can thus prune away many equivalent solutions where two empty lockages are present before all ships have been handled. Note that because the number of non-empty lockages in the optimal solution is not known, we allow for two or more lockages to be empty if all subsequent lockages are also empty. We can impose this by introducing the following constraints:

$$\sum_{s \in \mathcal{S}} \left(z_{s,l,k} + z_{s,l,k-1} \right) \geq \frac{1}{S} \sum_{\substack{s \in \mathcal{S} \\ \kappa>k}} z_{s,l,\kappa} \quad \forall l \in \mathcal{L}, k \in \mathcal{K} \setminus \{1\} \tag{4.48}$$

A second set of inequalities expresses the intuitively obvious statement that once a ship has arrived at one of the locks, it will enter the first available non-full lockage in the appropriate direction. Note that this does not imply that a ship must move at a high speed so that it arrives on time to enter the first following non-full lockage, but rather that if ship s, travelling e.g. in the upstream direction, has passed lock l and there is a non-full lockage k for lock $l+1$ that does not contain s, then $\bar{v}_s, l$ must be such that s has not yet arrived at lock $l+1$ at time $t_{l+1,k}$.

The mathematical formulation is as follows:

$$1 - \frac{\sum_{\sigma \in \mathcal{S}} z_{\sigma,l,k}}{C_l} - b_{s,l,k} - \sum_{\kappa<k} z_{s,l,\kappa} <= 0 \quad \forall l \in \mathcal{L}, s \in \mathcal{D}, k \in \mathcal{K}$$

$$1 - \frac{\sum_{\sigma \in \mathcal{S}} z_{\sigma,l,k}}{C_l} - b_{s,l,k} - \sum_{\kappa<k} z_{s,l,\kappa} <= 0 \quad \forall l \in \mathcal{L}, s \in \mathcal{U}, k \in \mathcal{K}$$

Finally, we introduce additional constraints on the $a_{s,l}$ and c_s variables:

$$\begin{aligned} a_{s,l} &\geq a_{s,l+1} + P_{l+1} + S_l \bar{v}_{s,l} && \forall s \in \mathcal{U}, l \in \mathcal{L} \setminus \{L\} \\ a_{s,l} &\geq a_{s,l-1} + P_{l-1} + S_{l-1} \bar{v}_{s,l-1} && \forall s \in \mathcal{D}, l \in \mathcal{L} \setminus \{1\} \\ c_s &\geq a_{s,1} + P_1 && \forall s \in \mathcal{U} \\ c_s &\geq a_{s,L} + P_L && \forall s \in \mathcal{D} \end{aligned}$$

Adding these valid inequalities to the reformulated LB model improves the performance of the model; we refer to Section 4.6 for computational experiments that compare the LB and LB+ models.

4.5 A single-lock based heuristic

While the MIP models allow to find a provably optimal solution, the computational burden can be difficult to overcome for larger instances. To investigate whether the use of an integrated MIP model for multiple locks is justifiable, we compare the exact solution to the solution obtained from an intuitive heuristic.

The heuristic solution is based on a decentralized approach where optimal decisions are made for the individual locks. To obtain an optimal solution to the single-lock problem in polynomial time, we use the results described in Chapter 2. From each individual solution, arrival times are then obtained for the neighbouring locks. Each of the locks is iteratively scheduled to minimize flow time individually, using only the known ship arrival times. We are, however, facing the problem that with the locks arranged in sequence, the arrival time of a ship at a lock is known only once the schedule of the previous lock that processes this ship is known. Thus, initially, only the downstream arrivals for the first lock and the upstream arrivals for the last lock are known.

We resolve this problem by iteratively scheduling each of the single locks for only those arrival times that are known. We then update arrival information for the following iterations and repeat the process until the solution has converged. A solution has converged when the arrival times computed in the current iteration are equal to the arrival times of the previous iteration, for each lock. An outline of this procedure in pseudo-code is given in Algorithm 2, where SLS1(U, D) represents the single lock solver procedure that returns a solution to the single lock problem given a set of upstream arrivals U and downstream arrivals D, and f_u (f_d) is a function that computes the upstream (downstream) arrival times at a lock given from a given schedule at the previous lock. The input for this procedure consists of the reciprocal of the (identical) ship speed, section lengths, and sets with all arrival times of ships travelling upstream, as well as downstream, at the lock where they enter the system. These sets of arrival times will be denoted by U_1^L and D_1^1 respectively. The heuristic is named RISL, short for 'repeated iterations of single lock scheduling'.

Input: $\mathcal{U}$, $\mathcal{D}$, S_l (for $l \in \mathcal{L} \setminus \{L\}$), $\bar{v}$
$i \leftarrow 1$
$U_1^L \leftarrow \mathcal{U}$, $D_1^1 \leftarrow \mathcal{D}$
repeat
  $U_{i+1}^L \leftarrow U_i^L$, $D_{i+1}^1 \leftarrow D_i^1$
  for $l \in \mathcal{L} \setminus \{1\}$ **do**
    $U_{i+1}^{l-1} = f_u(\text{SLS1}(U_i^l, D_i^l)) + \bar{v}S_{l-1}$
  for $l \in \mathcal{L} \setminus \{L\}$ **do**
    $D_{i+1}^{l+1} = f_d(\text{SLS1}(U_i^l, D_i^l)) + \bar{v}S_l$
  $i \leftarrow i + 1$
until $\left(U_i^l = U_{i-1}^l \textbf{ and } D_i^l = D_{i-1}^l \textbf{ for all } l \in \mathcal{L}\right)$ **or** $i = 10L$

Algorithm 2: Pseudo-code algorithm for the RISL heuristic

As an example, consider for a moment that we have only two locks. We start by scheduling the first lock while only considering those ships for which the arrival time is known. Next, we schedule the second lock, including only the initially available arrival information. After obtaining a schedule for the second lock, we update the arrival times at each of the locks based on the individual schedules for both locks, adjusted with the appropriate lockage duration and travel times. We now recalculate the individual schedule for each of the locks and keep repeating this procedure until the solution has reached convergence or a fixed iteration limit has

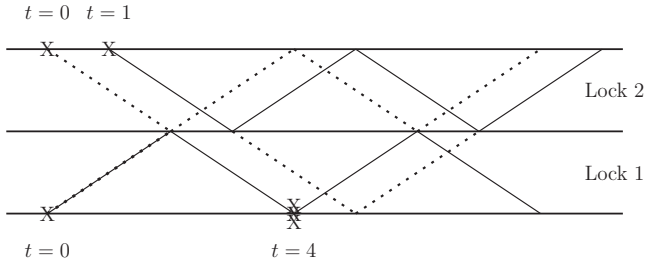


Figure 4.6: Instance for which RISL does not converge. The locks have $T_1 = T_2 = 2$ and $C_1 = C_2 = \infty$. The dotted lines represent an optimum solution for each lock after each even iteration; the solid lines represent an optimum solution for each lock after each odd iteration.

been reached. We note that this iteration limit is required to ensure that the procedure terminates, since it cannot be guaranteed that the procedure converges for every input. In fact, it is possible to construct an instance so that the algorithm cycles infinitely between two solutions; such an instance is illustrated in Figure 4.6.

Note that the RISL procedure does not correspond entirely to the decisions made in practice. Because the arrival times are obtained from previous iterations, each lock operator thus implicitly takes future decisions of neighbouring locks into account. Since this information is not available in practice, we may expect the iterative procedure to perform better than a typical decentralized solution in practice. For an estimation of the waiting time that can be avoided by scheduling the system of locks as a whole as opposed to the decentralised scheduling of the individual locks, see Section 4.6.2; there, computational experiments compare solutions obtained with a MIP model to solutions obtained with the RISL heuristic.

Notice that when deadlines are imposed, for example when considering the emission objective, deadlines should be specified for each of the single-lock instances to obtain sensible results. Since it is unclear how to derive deadlines at each of the individual locks while still allowing the model sufficient freedom to minimize emissions, we do not consider the emission objective for the RISL heuristic. In Section 4.6.2, we restrict ourselves to the flow time objective.

4.6 Computational study

In this section, we evaluate the performance, with respect to minimizing total flow time, of the two models and their respective performance improvements described in Section 4.4. We then use the lockage-based model to minimize the emissions and evaluate the trade-off between emissions and flow time on instances based on real-world data. All instances and results are made available online at `https://perswww.kuleuven.be/~u0086328/lockmasterdata.html`.

4.6.1 Comparison of the MIP-models

We implement the different model formulations with the flow time objective in CPLEX (version 12.5.1). We evaluate the performance on different sets of instances. Our reference scenario consists of instances with 3 identical locks and 15 ships. The lockage time is equal to 12 minutes. Arrival times are generated randomly from a Poisson arrival process with parameter $\lambda = 0.0536$. These value are derived from a real-world dataset, see also Section 4.6.3. The Poisson parameter corresponds to an expected time between arrivals of 18.66 minutes. For simplicity, the distance between the locks is considered to be zero. This allows us to ignore the minimum and maximum speed. Results concerning the distance and ship speed between locks are presented in Section 4.6.3.

We consider different values for the time unit in order to investigate the impact on solution performance. In the reference scenario, we assume a time unit of 5 minutes, i.e. all arrival times and the lockage time are rounded to the nearest multiple of 5 minutes; the lockage duration for each lock is thus assumed to be 2 time units. Other values for the time unit are considered in what follows.

Since we are minimizing the flow time, we need not impose any deadlines. A summary of the input parameters for the reference scenario is given in Table 4.1. Each scenario is evaluated for 10 randomly generated instances. We enforce a time limit of 15 minutes per instance. No model tuning is required, and the CPLEX solver options are left to their default values.

For the reference scenario, we compare the performance of the models presented in Section 4.3 as well as the models with the performance improvements described in Section 4.4. The time-indexed and lockage based models, including the FCFS constraints discussed in Section 4.3.3, are denoted by TI and LB respectively, while the models with the improvements described in Section 4.4 are labelled TI+ and LB+. Table 4.2 presents the

Input parameters			
$S =$	15	$C_l =$	2
$L =$	3	$\lambda =$	0.0536
$S_l =$	0	$D_s =$	∞
$P_l =$	2	$[V_s^{\min}, V_s^{\max}] =$	[2, 12]

Table 4.1: Instance properties for the reference scenario

# instances solved			
TI	TI+	LB	LB+
10	10	0	10
avg. relative optimality gap			
TI	TI+	LB	LB+
0 %	0 %	185%	0%

time to generate model [s]			
TI	TI+	LB	LB+
21.14	1.81	2.63	1.44
solution time [s]			
TI	TI+	LB	LB+
8.89	0.49	900.15	12.89

Table 4.2: Model comparison for the reference scenario

results.

Perhaps surprisingly, the LB models do not perform well despite having significantly less variables than the TI model. To illustrate, the average number of variables and constraints in each of the models is presented in Table 4.3. Due to the filtering of variables in the TI+ model, this model has an even lower number of variables, although this number depends on the size of the time unit as well as the value $\overline{T}$, as discussed in Section 4.3. While the LB model performs very badly, the stronger LB+ model does outperform the TI model without preprocessing. This is good to note since the TI preprocessing is only possible when minimizing the flow time, or alternatively when ship speed is fixed, whereas the improvements introduced in the LB+ model can also be applied in the emission context with variable speed.

For both models presented in Section 4.3, it can be seen that the modifications suggested in Section 4.4 provide a strong reduction in computation time. In what follows, we investigate the sensitivity of the models' performance with respect to different input parameters. Given the drastic improvement of the TI+ and LB+ models over their basic variants, we limit ourselves to the improved models only.

# variables (binary, continuous)			
TI	TI+	LB	LB+
(3053, 1)	(784, 1)	(1350, 106)	(1395, 1549)
# constraints			
TI	TI+	LB	LB+
137262	17921	48253	10275

Table 4.3: Number of variables and constraints for reference instances

Instance size

We increase the size of the reference instances up to 25 and 35 ships. The other input parameters remain unchanged. Table 4.4 presents the results for the TI+ and LB+ model. As a consequence of our method of generating Poisson arrivals, the time horizon is expected to grow proportionally with the number of ships. For the TI model, this implies that the number of variables (and constraints accordingly) will increase. Although the number of variables in the LB model does not depend on $\overline{T}$, several big-M values in the LB+ constraints will increase as a result. The results indicate that the TI+ continues to outperform the LB+ model even in this case. Notice that the time required to generate the TI+ model grows much faster than that of the LB+ model, which reflects the rapidly increasing number of variables. This is also an early indication of the fact that the time-indexed models will easily fail to generate larger models due to insufficient memory, whereas the lockage-based models do not suffer from this problem but fail to find high-quality solutions. Note that not many meaningful conclusions can be drawn from the reported computation time for the LB+ model. Indeed, only a single instances is solved to optimality in the setting with 25 ships, and no instances are solved for the setting with 35 ships. Repeating these tests with an increased time limit of 2 hours shows that, in the setting with 25 ships, 8 instances can be solved to optimality; the average computation time for the solved instances then equals 4116.9 seconds, a sizeable increase compared to the reference scenario with 15 ships.

Number of locks

To investigate the effect of the number of locks, we solve the reference scenario for a single lock as well as for a set of 5 locks. For the latter scenario, we maintain the assumption that all distances are equal to zero.

# instances solved	
TI+	LB+
10	1
avg. relative optimality gap	
TI+	LB+
0 %	18 %
time to generate model [s]	
TI+	LB+
7.62	5.33
solution time [s]	
TI+	LB+
2.39	883.19

# instances solved	
TI+	LB+
10	0
avg. relative optimality gap	
TI+	LB+
0 %	67%
time to generate model [s]	
TI+	LB+
22.58	7.02
solution time [s]	
TI+	LB+
9.11	900.62

Table 4.4: Results for varying problem sizes: 25 ships (left) and 35 ships (right)

We note that the single-lock scenario should be considered as a simple reference benchmark only, since the methods described in Chapter 2 can be used to obtain an optimal solution in polynomial time, which outperforms the MIP approach presented here. The results for both sets of instances can be found in Table 4.5.

For a single lock, it can be seen that the performance of both models is comparable. In the single lock setting, the set of big-M constraints (4.43) and (4.44) vanish from the model, strengthening the LP relaxation. When increasing the number of locks, however, the opposite occurs and additional big-M constraints are added. As a consequence, the LB+ becomes two orders of magnitude slower than the TI+ model.

Time unit

Here, we vary the size of the time unit, i.e. the discretisation step size. We solve instances with a one minute and with a 15 minute time unit. All original input parameters are rescaled and rounded to the nearest unit, e.g. in the 15 minute case, the lockage duration $P_l = 1$ and all arrival times A_s are rounded to the nearest multiple of 15 minutes. Recall that the time unit equalled five minutes in the reference scenario. Obviously, the computation

# instances solved	
TI+	LB+
10	10
time to generate model [s]	
TI+	LB+
1.24	0.97
solution time [s]	
TI+	LB+
0.17	0.46

# instances solved	
TI+	LB+
10	10
time to generate model [s]	
TI+	LB+
2.40	1.22
solution time [s]	
TI+	LB+
1.17	168.39

Table 4.5: Results for varying number of locks: single lock (left) and 5 locks (right)

time for the TI+ model is expected to increase significantly for the one minute scenario, as the number of variables increases. However, a negative effect on the performance of the LB+ model is also expected with an increasing value for $\overline{T}$ and the related big-M values in the constraints. Table 4.6 shows the results. We note that the reported results relate only to the performance of solving the instances. We assume that the arrival times are rounded to the nearest multiple of the time unit, and thus do not assess the waiting time that would occur if ship arrivals occur in between the rounded times at which a lockage is allowed to start.

Unsurprisingly, both models perform better when a larger time unit is chosen and the computation time for the TI+ model increases significantly compared to the reference scenario. The LB+ model, on the other hand, experiences only a minor increase in computation time for a smaller time unit. While the performance of the LB+ model suffers significantly from increasing the number of variables, for example by increasing the number of ships or locks in an instance, changing only the value of $\overline{T}$ is not as detrimental to performance.

Arrival rate

We do not perform a separate test to check for the influence of the arrival rate parameter λ, since varying this parameter is equivalent to varying the time unit (discussed above) and simultaneously rescaling the value for the lockage duration so that the ratio of lockage duration to time unit

# instances solved	
TI+	LB+
10	10
time to generate model [s]	
TI+	LB+
5.69	1.00
solution time [s]	
TI+	LB+
9.13	17.91

# instances solved	
TI+	LB+
10	10
time to generate model [s]	
TI+	LB+
1.15	1.24
solution time [s]	
TI+	LB+
0.16	5.71

Table 4.6: Results for varying time unit size: 1 minute (left) and 15 minutes (right)

remains unchanged. Results similar to the scenarios for the time unit can be expected, where a low arrival rate favours the LB+ model, and a relatively high arrival rate favours the TI+ model.

Summary of the MIP comparison

To summarize, we can conclude that the TI+ model outperforms all other models for the tested instances. However, the model does suffer from two important drawbacks:

- As with any time-indexed model, the number of variables depends on the length of the time horizon. The computation time increases particularly fast when either choosing a small time unit (e.g. minutes) or for low arrival rates.
- Since the preprocessing assumes that the ship speed is known, the number of variables can only be reduced when speed is assumed to be fixed or when the objective is to minimize the total flow time.

The first item is illustrated by the tests performed for 15 ships and 3 locks with a time unit of 1 minute. For these instances, the TI+ and LB+ models have comparable performance, indicating that there are 'break-even' input parameters beyond which the LB+ model is to be preferred over the TI+ model.

Especially the second item justifies the use of the LB+ model when considering emissions or investigating the impact of variable speed, since

input		
S	=	15
L	=	5
S_l	=	0
P_l	=	12
C_l	=	2
λ	=	0.107
D_s	=	∞

Table 4.7: Input for the RISL heuristic.

Total flow time [min]		
TI+	RISL	Relative gain
1106	1179	6.19 %
1155	1155	0 %
1030	1030	0 %
1066	1066	0 %
1022	1044	2.11 %
1080	1080	0 %
1225	1249	1.92 %
1056	1057	0.09 %
1073	1101	2.54 %
1022	1115	8.34 %
Average computation time [s]		
TI+	RISL	
30.59	0.40	

Table 4.8: Results for instances comparing the TI+ and RISL solutions.

the disadvantages of a time-indexed formulation are much more prominent in the TI model, which is outperformed by the LB+ model even in the reference scenario.

4.6.2 Comparing MIP and heuristic solutions

We compare the solution quality of centralized solutions, obtained with the TI+ model, with that of the RISL heuristic described in Section 4.5. The input used for the instances is summarized in Table 4.7. We again compare the results for 10 instances; the results are listed in Table 4.8. The heuristic finds an optimum solution in 4 out of the 10 instances. The average performance gain by centralizing the scheduling is 2.12%, although the variation over the instances is quite large, up to a maximum of 8.34%. Naturally, this comes at the expense of a significantly increased computation time. With respect to minimizing the total flow time, we conclude that there is value in obtaining a centralized schedule. Also note that the emission objective inherently requires an integrated approach.

4.6.3 Evaluating emission reductions

Instance data

We now evaluate the trade-off between emission reduction and flow time reduction based on an instance created from real-world data. We focus on a section of the upper Scheldt river near Ghent, Belgium. We consider three locks, each consisting of a single chamber, separated by a distance of 10.9 km and 12 km respectively. Arrival times are generated randomly for 25 ships, with a time unit of one minute. Based on a dataset with historical arrival data provided by 'Waterwegen en Zeekanaal', the organisation responsible for managing the locks, an arrival rate of $\lambda = 0.0536$ can be estimated between 6 a.m. and 8 p.m. We use this arrival rate, noting that we do not fix the time horizon of the instances since this is determined from the Poisson arrival process where both the number of arrivals and the parameter Poisson parameter λ are fixed. We report the average results obtained over 10 instances.

In order to derive the piecewise linear function approximating $\bar{E}(\bar{v})$ we employ a tool originating from the ARTEMIS project, see Boulter (2007). This tool allows to evaluate the CO_2 emissions as a function of multiple inputs, including the ship speed and waterway characteristics. The tool's default ship data is used for ships of the type 'RHK', named after the Rhine-Herne Canal for which ships of this size were originally designed. The RHK ships have a size which is representative for freight ships that pass through the upper Scheldt river. The waterway characteristics entered in the ARTEMIS tool are a water depth of 3 m and an average waterway width of 50 m.

Using the tool, we evaluate the CO_2 emissions at a speed of 2, 5.33, 8.66, and 12 km/h, depicted as dots in Figures 4.7 and 4.8. We transform the results into a piecewise linear approximation of $\bar{E}(\bar{v})$ with three linear segments. It is important to note that if a piecewise linear function is translated into a function of emission depending on speed, the result is not convex. This is illustrated in Figure 4.7. The effect can be made arbitrarily small, however, by choosing the piecewise linear segments sufficiently short. The resulting functions $\bar{E}(\bar{v})$ and $E(v)$ are shown in Figures 4.7 and 4.8 respectively. Further, we impose a minimum speed of 2 km/h and the canal's maximum speed of 12 km/h for all ships.

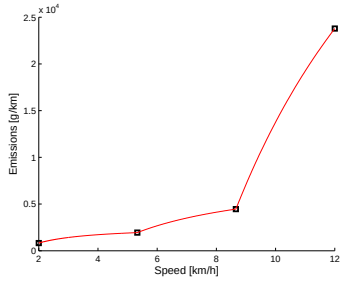


Figure 4.7: Resulting approximation of $E(v)$.

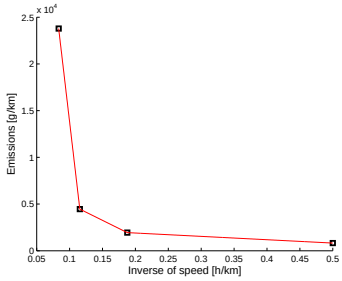


Figure 4.8: Piecewise approximation of $\bar{E}(\bar{v})$.

Trade-off between emissions and flow time

The trade-off between flow time and emissions is evaluated as follows: first, the LB+ model is used to minimize the total flow time without imposing any deadlines, where ship speed is fixed to the maximum allowed speed of 12 km/h. Notice that the flow time of each ship is at least equal to 150.5 min: the time needed to traverse the entire canal at maximum speed plus the total time spent inside locks. To speed up the computation, we set a deadline $D_s = A_s + 240$ min for each of the ships. All instances are solved to optimality.

Solving these instances yields a set of completion times, which will be used as the reference deadlines in what follows. Next, we treat the ship speed as variable and gradually relax the deadlines so that the flow time for each individual ship is allowed to increase by a given percentage relative to the reference scenario. In each of these modified instances, the ship speed is variable. Table 4.9 lists the average of the total emission value over the instances as the flow time is increased. Note that a reduction in total emission is possible even without relaxing the reference deadlines, as the variable ship speed allows the slowing down, thus reducing emissions, when waiting time is unavoidable at the locks. This scenario, where the reference deadlines are imposed while ship speed is variable, will be referred to as the reference scenario. Note that the reference scenario allows an average emission reduction of 3.44% over the initial setting with fixed speed, while the total flow time remains unchanged. Table 4.9 also lists the average computation time as flow times are increased. We should note that determining the initial completion times with fixed speed had in

flow time increase [%]	CO_2 [tonnes]	CO_2 reduction [%]	comp. time [s]
0 (12km/h)	13.616	-	5.13
0 (ref)	13.148	0	4.84
1	12.816	2.5	4.85
5	11.534	12.3	5.01
10	9.702	26.2	5.31
20	5.861	55.4	18.53
30	2.788	78.8	37.58
40	2.289	82.6	97.58
50	2.064	84.3	151.13
60	1.834	86.1	243.22

Table 4.9: Averaged emission results. The first row refers to the setting where ship speed is fixed at $V_s^{\max}$. In all other rows, the ship speed is variable and the deadlines are adjusted to constrain the increase in flow time.

fact the largest computation time. On average, this required 895 seconds, although the required time varied greatly by instance, with a minimum of 71 seconds and a maximum of 4374 seconds. Once the reference deadlines times are determined, subsequent tests complete much faster since the deadlines strongly constrain the feasible solution space.

Figure 4.9 shows the trade-off between emissions and flow times, averaged over the 10 instances considered. The emission reduction is given as a percentage relative to the solution which minimizes the total flow time where the ship speed is variable, i.e. the reference scenario. It can be seen that the trade-off is approximately linear up to a point where the curve flattens. This is to be expected from the relationship between emissions and speed, where a much steeper slope is found at the highest speed values.

It should be mentioned that the trade-off is generally slightly more erratic for a single instance. When the increase in flow time is insufficient to allow any changes to the assignment of ships to lockages, an emission reduction is possible only by slightly delaying the starting time of the lockages. However, when the increase in flow time becomes sufficiently large, the assignment of a ship may change to a later lockage, introducing a discontinuity in the trade-off curve. Figure 4.10 shows the trade-off profile evaluated for a single instance. While this effect is barely visible in this case, it is expected to be more prominent when distances S_l are

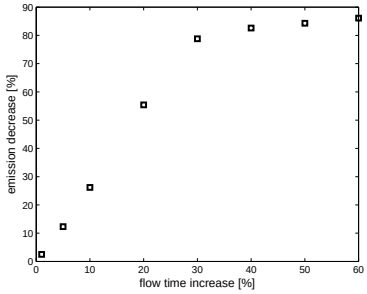


Figure 4.9: Trade-off between flow time and emissions, averaged.

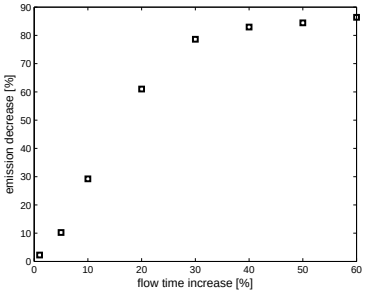


Figure 4.10: Trade-off between flow time and emissions, single instance.

smaller, or for a larger lockage duration P_l.

4.7 Conclusion

In this chapter, we described two mixed integer programming models which allow to minimize the total flow time for multiple locks arranged in a sequence. Both models can be easily extended to include additional constraints or alternative objective functions such as minimizing the total greenhouse gas emissions, as opposed to the total flow time. Computational testing confirms that a time-indexed model performs well when the unit of time is chosen sufficiently large when the objective is to minimize the total flow time. This model permits a preprocessing step where the number of variables is reduced, greatly benefiting the performance. Perhaps more surprisingly, a second model where the number of variables does not depend on the time unit does not seem to outperform the first model, even with the addition of valid inequalities to tighten the LP-relaxation. In the context of minimizing emissions, however, the latter model will easily outperform the former, since the preprocessing is then no longer possible. Comparing the exact solutions to a heuristic indicates that, while the heuristic may frequently find an optimal solution, significant improvements can be achieved for other instances. It is also shown that for the proposed heuristic, which is based on iteratively scheduling the individual locks to optimality, convergence cannot be guaranteed. Different computational experiments evaluated the trade-off between emission reduction and increase in flow time for a system of three locks on the upper Scheldt river.

Compared to the fixed speed scenario, an emission reduction of 3.44% could be achieved with no impact on the total flow time. Relaxing the flow time for individual ships allows a further reduction at an approximately linear rate.

As a topic for future research a number of practical extensions, similar to those presented for a single lock in Chapter 2, could be considered. Further, the limited performance for larger instances suggests a decomposition approach such as branch and price, which may overcome the weak LP relaxation of the LB model while avoiding the inclusion of many unnecessary variables and suffering from the same drawback as the TI model. A different approach would be to initialize the solution procedure with a known feasible solution, such as the RISL heuristic based on the algorithm for single locks.

Chapter 5

No-wait scheduling for parallel chambers[1]

This chapter explores the scheduling of a lock that consists of multiple parallel chambers. More specifically, we are interested in so-called no-wait schedules, i.e. schedules where none of the ships incurs any waiting time. The resulting problem closely resembles an interval scheduling problem. A formal problem description for the general problem is provided in Section 5.1. In proving results for various special cases of this general problem, a number of different results from literature can be improved upon. The related literature and an overview of the relevant results obtained in this chapter are discussed in Section 5.2; a summary is given in Table 5.1. Different special cases of the general problem with parallel chambers are then investigated. Section 5.4 considers the uni-directional setting for a lock with two arbitrary chambers. Section 5.5 concerns the more general bi-directional setting. In a different special case, all chambers are identical; this setting is covered in Section 5.6. Finally, we look at the more general setting with arbitrary chambers. Section 5.7 provides a dynamic programming algorithm, while Section 5.8 shows that this general problem is NP-complete.

The contributions of this chapter are the following. We show how our problem relates to known interval scheduling problems, as well as

[1]The research presented in this chapter, formatted as a journal article, has been submitted for publication and is currently undergoing peer review. An electronic version is available as a research report, see Passchyn et al. (2016b).

to a particular graph colouring problem on multiple unit interval graphs. We explore the relationships between these problems and discuss the complexity of different problem variants. In particular, for a lock consisting of two chambers we are able to characterize the feasible instances and use this result to obtain efficient algorithms: the bi-directional setting can be solved in $O(n^2)$ time, whereas the uni-directional setting can be solved in $O(n)$ time. We also provide a linear-time algorithm for the special case with identical lock chambers. Furthermore, we describe an $O(mn^m)$ dynamic programming approach for the more general case with arbitrary chambers, and prove that the problem is strongly NP-complete when the number of chambers is part of the input.

5.1 Problem definition

We consider a single lock that consists of m parallel chambers. Since there is only a single lock, we will ignore the index for the lock throughout this chapter. Let $\mathcal{C}$ denote the set of chambers for this lock. The lockage duration and capacity of a chamber $c \in \mathcal{C}$ are thus identified by T_c and C_c respectively. The arrival time of a ship s in the set of ships $\mathcal{S}$ is denoted by A_s. For convenience, we also define $\mathcal{A}$ as the set of arrival times, i.e. $\mathcal{A} = \{A_s \mid s \in \mathcal{S}\}$.

Our interest here is exclusively on the existence of so-called *no-wait* schedules. A no-wait schedule is a schedule where each ship, upon its arrival, can enter a chamber of the lock immediately. Thus, in a no-wait schedule, each ship $s \in \mathcal{S}$ leaves the lock at either $A_s + T_1$, $A_s + T_2$, ..., or $A_s + T_m$, depending on the particular chamber to which the ship is assigned. We say that an assignment of ships to lock chambers is feasible if each ship is served by a chamber upon its time of arrival and if, for each chamber, the performed lockages do not overlap and alternate between the upwards and downwards direction. Notice that empty lockages may be required to return a chamber to the position where a ship arrives. The question we address is thus: given the arrival time and the direction of travel for each ship, does there exist a feasible assignment of each ship to a chamber such that no ship has to wait; more compactly: does there exist a no-wait schedule?

We are aware that, from a practical point of view, this problem description does not include all relevant features such as the size of a ship and ship-dependent lockage durations. However, in order to be able to solve these practical problems, it is good to understand the behaviour of

this more basic problem. The following also justifies the focus on no-wait schedules initially: as mentioned in Section 1.1, large vessels may be subject to a 'tidal window' in certain ports, i.e. they are able to enter the port only during a limited time interval. For this reason, a policy can be enforced that sea-going vessels do not incur any waiting time at the locks that allow entry into the port area. A similar argument can be made for locks on inland waterways when serving certain classes of ships carrying dangerous cargo. Locks must then be scheduled such that these ships are immediately served by the lock upon their time of arrival.

Moreover, as we argue below, the problem can be seen as a particular interval scheduling problem, which is of independent interest, and some of our results have implications for other interval scheduling problems. A special case of the problem that is of interest is the uni-directional setting.

5.1.1 A note on sorted arrival times

Some of the results discussed in this chapter apply exclusively to sorted input, i.e. these results require the assumption that arrival times are given in non-decreasing order. Due to the well-known $\Omega(n \log n)$ lower bound on comparison-based sorting algorithms, it may not be possible to improve beyond a complexity of $O(n \log n)$ in the general case of unsorted arrival times. However, assuming that the arrival times are known in sorted order may be justified. For instance, when iteratively applying the presented methods, a large part of the input may have remained unchanged since earlier iterations, so that sorting is no longer required for a large subset of the given arrival times. Furthermore, the input may be unsorted but may satisfy additional assumptions which allow the use of a non-comparison-based sorting algorithm such as radix sort, which runs in linear time. Therefore, we choose to report the complexity assuming that the arrival times are sorted.

5.2 Relation to literature

As briefly mentioned in the literature overview of Section 1.2, the problem of scheduling locks consisting of parallel chambers remains largely unexplored. The only work, as far as we are aware, that deals with exact methods for this setting is Verstichel et al. (2014b), who present a MIP model where the chamber assignment decision is a part of an integrated approach for the packing and scheduling of ships and lock chambers. To the best of our

knowledge, the computational complexity of scheduling locks with multiple chambers has not been directly addressed in the literature. Below, we discuss two problems that can be related to the no-wait scheduling problem for a lock with parallel chambers. We refer to Table 5.1 in Section 5.3 for an overview of the different results presented in this chapter, and the related results from literature which are extended.

5.2.1 Interval scheduling

A closely related problem in the context of scheduling parallel machines is interval scheduling, which can be stated as follows. Given are a set $\mathcal{J}$ of jobs, each represented by an interval $[s_j, f_j)$ for $j \in \mathcal{J}$. Additionally, there are a number of identical machines. Each job requires processing on one of the machines during its stated interval, in such a way that no machine processes two jobs of which the intervals intersect. The problem consists of finding a schedule that minimizes the number of machines required to process all jobs.

We can phrase our problem setting introduced above in terms of interval scheduling as follows. Let a chamber be a *machine*, and let a ship be a *job*. Furthermore, let the direction of travel for each ship correspond to a *job type* for each job. Multiple intervals are associated to each job, one for each machine in the instance. The starting time of each of the intervals corresponding to a particular job i is equal to R_i; the ending times of these intervals are given by $R_i + P_j$, $j \in \{1, \ldots, m\}$. Clearly, the starting time R_i of a job i corresponds to the arrival time A_s of a ship s, and the processing time P_j corresponds to the lockage duration T_c of a chamber c. Notice that when considering a particular interval, it is associated with a job and with a machine. A feasible solution consists of a selection of intervals such that (i) one interval corresponding to each job is selected, (ii) the selected intervals that correspond to a machine are disjoint, and even more: when two consecutive intervals of a machine correspond to jobs with the same job type, there must be a difference of P_j between the ending point of the earlier interval and the starting point of the later interval. The requirement involving this difference is needed because a chamber transporting a ship needs P_j time units to return before transporting another ship that travels in the same direction. Notice that in the uni-directional case (when all jobs have the same type), this difference requirement vanishes since it can be modelled by assuming that the length of the intervals equal $2P_j$.

Interval scheduling is a well-studied subject, see Kolen et al. (2007) for

an overview. A recent paper by Krumke et al. (2011) deals with interval scheduling on related machines, which can be formulated as follows. Given are m machines, each with a certain speed V_j $(1 \leq j \leq m)$, and n intervals specified by a starting point R_i and a processing time P_i $(1 \leq i \leq n)$. They show that even deciding the existence of a schedule is NP-complete. Clearly, this setting is relevant for our problem, certainly when we consider the uni-directional variant of our problem: by setting the speed V_j of machine j equal to ${}^{T_j}/_{T_{\max}}$ (where $T_{\max}$ refers to the maximum lockage time over the chambers, i.e. $T_{\max} = \max_j T_j$), the two problems are very much related. In fact, our problem is a special case of the problem in Krumke et al. (2011), since in our case, the lengths of intervals corresponding to a particular machine (the processing requirements) are identical.

Another interesting paper, by Böhmová et al. (2013), concerns a version with machine-dependent intervals. Here, a job corresponds to a set of intervals, one for each machine, and to schedule a job exactly one of its intervals must be selected. A set of selected intervals is then called feasible if the intervals corresponding to the same machine do not overlap. In their paper, they consider different special cases, one of which is of primary importance to our problem: the problem with so-called *cores*, where all intervals corresponding to the same job have a point in time in common. More specifically, Böhmová et al. (2013) deal with the problem where all intervals of a job end at the same time; they prove that deciding whether a feasible selection of intervals scheduling all jobs exists is NP-complete, solving an open problem from Sung and Vlach (2005). As this problem is equivalent to dealing with the problem where all intervals of a job start at the same time, this seems to be identical to our problem. There is one difference however: in our case, the set of lengths of the intervals that correspond to a job is the same for all jobs, which is not necessarily the case in Böhmová et al. (2013). They also mention a dynamic program given in Sung and Vlach (2005); translated in our terms, this means that the uni-directional case can be solved by a dynamic program in $O(mn^{m+1})$ time. In addition, they mention that the uni-directional case with two machines is polynomially solvable by a reduction to 2-SAT.

5.2.2 Graph colouring

Another related problem concerns the colouring of graphs. The well-known graph colouring problem can be stated as follows. Given a graph $G = (V, E)$, decide whether there exists a partition of V into k subsets so that, for every edge $(i, j) \in E$, vertices i and j are not contained in the

same partition. Equivalently, decide whether there exists a partition of V into k subsets such that each subset is an independent set in G.

Now consider the following graph colouring problem on a number of unit interval graphs with a common node set. For each unit interval graph, by definition, a set of equal-length real intervals exists where each interval corresponds to a node and an edge exists if and only if the intervals corresponding to these nodes overlap. Given are m unit interval graphs (V, E_1), ..., (V, E_m) where the edge sets satisfy $E_1 \subseteq \ldots \subseteq E_m$. Notice that these m graphs have the node set V in common. The question is: does there exist a partition of the node set V into m subsets $V_1, \ldots, V_m$ such that $V_j \subseteq V$ is an independent set in (V, E_j) for each $j \in \{1, \ldots, m\}$?

The two problem descriptions are related as follows. Consider the uni-directional case of the problem formulated in the "lock"-description and assume that all arrival times are distinct. Build a node set V by having a node in V for each arrival of a ship. Next, build an edge set E_j by having an edge in E_j between the nodes $i_1, i_2 \in V$ if and only if $A_{i_1} + 2T_j > A_{i_2}$ for $j = 1, \ldots, m$. It is easily seen that if chamber j is used to serve ship i_1, it can return to the position where ships arrive no earlier than $A_{i_1} + 2T_j$; for each pair of nodes (i_1, i_2) for which an edge exists in edge set E_j, chamber j can thus not be used to serve both ships in any no-wait solution. Observe that if the lock chambers are ordered such that $T_1 \leq \ldots \leq T_m$, this immediately implies that $E_1 \subseteq \ldots \subseteq E_m$. The existence of a partition of V into subsets V_j, with V_j an independent set in (V, E_j) for all $j \in \{1, \ldots, m\}$, then corresponds to the existence of a no-wait schedule. Indeed, the nodes, or arrivals, in V_j can be handled by chamber j in a corresponding solution to the "lock" problem.

Observe that the "lock"-description gives rise to a problem that is more general than the problem from the "graph"-description in two ways: (i) multiple ships may arrive at the same time, and (ii) ships may travel in the downstream as well as the upstream direction. One point to note is that to obtain a "lock"-instance from a given "graph"-instance, an interval representation (leading to the values A_s) needs to be determined. In this work, we do not not consider this conversion explicitly.

5.3 Notation and summary of results

Throughout the remainder of this chapter, we use the "lock" description of the problem to describe our results. Thus, our results are formulated in terms of locks and ships; for example, the phrase 'assigning ship s to

chamber c' corresponds to 'executing job s on machine c' in the "interval" description, and to 'placing node s in subset V_c' in the "graph" description. For convenience, we introduce a notation to refer to the special cases of NLS discussed in what follows. We refer to the problem settings as NLS{-uni, $\emptyset$}{-2, -m, $\emptyset$}{-id, $\emptyset$}{-distinct, $\emptyset$}, where

- *uni* refers to the uni-directional case (where omitted, bi-directional traffic is implied),
- *2* refers to the setting with two lock chambers and m refers to the setting where the number of chambers m is fixed (omitting these implies that the number of chambers is part of the input),
- *id* refers to the setting with identical chambers (omitting this implies that values for lockage time and capacity are arbitrary),
- *distinct* refers to the setting where all arrival times are distinct (omitting this implies that multiple ships may arrive simultaneously).

We point out that specifying additional parameters in the problem name increasingly yields a more specific case of the problem. NLS, which describes the setting with an arbitrary number of non-identical chambers and bi-directional traffic, thus describes the most general problem covered by this notation.

Table 5.1 lists the notation for the different problem settings discussed in this chapter, and gives an overview of known results from literature and the new results presented in this chapter. Based on the chamber characteristics, we consider the following settings:

1. The setting where $m = 2$ with two arbitrary chambers. For the uni-directional setting, called NLS-uni-2, we give necessary and sufficient conditions for deciding on the existence of a no-wait schedule (Section 5.4), and we show how to find such a schedule in $O(n)$ time provided that the arrival times are sorted (see Section 5.1.1). Further, for the bi-directional case NLS-2, we give a reduction to 2-SAT that leads to an $O(n^2)$ algorithm (Section 5.5).

2. The setting where $C_c = C$ and $T_c = T$ for all $c \in \mathcal{C}$, i.e. the setting with m identical chambers. The resulting problems are called NLS-uni-id and NLS-id, and we show that we can solve these problems in $O(n)$ time for sorted arrival times (Section 5.6).

3. The setting where the number of chambers m is fixed. We give a dynamic program (DP) for our problem that runs in polynomial time (Section 5.7). For the problem of finding a feasible no-wait schedule described here, this DP strengthens the result in Sung and Vlach (2005) that can be applied to the uni-directional case.

4. The setting with an arbitrary number of chambers. We prove that the uni-directional case of this variant is NP-complete (Section 5.8). This result strengthens both the result given in Krumke et al. (2011) and a result in Böhmová et al. (2013).

5.4 Two arbitrary chambers, uni-directional

In this section we deal with the case of two chambers, more specifically with the uni-directional setting. In order to serve a ship with chamber $c \in \{1, 2\}$, a lockage duration T_c is incurred. We assume $T_1 \leq T_2$. We refer to the two chambers as the *short* and *long* chamber respectively, and to their lockages as *short* and *long* lockages.

In Section 5.4.1, we first look at problem NLS-uni-2-distinct, i.e. the special case with two locks where (i) all ships travel in the same direction, and (ii) all arrival times are distinct. Notice that the numbers C_1, C_2 are then irrelevant since a chamber contains at most one ship. In Sections 5.4.2 and 5.4.3, we extend our approach to a more general setting where some ships are pre-assigned to a chamber and describe how this result can be used to model the setting NLS-uni-2, where the numbers C_1 and C_2 become relevant.

5.4.1 Uni-directional setting, distinct arrival times

As argued in Section 5.1.1, we assume throughout this section that the arrival times are given in sorted order. We describe an $O(n)$ algorithm under this assumption. We organize this section as follows: in Section 5.4.1 we describe the construction of a graph corresponding to an instance of NLS-uni-2-distinct, and discuss some basic observations. We first prove a theorem characterizing feasibility, and then use this result to describe an $O(n)$ algorithm.

		2 arbitrary chambers	m identical chambers	m arbitrary chambers	m arbitrary chambers (m part of the input)
Uni-directional	notation	NLS-uni-2	NLS-uni-id	NLS-uni-m	NLS-uni
	known	$O(n^2)$ (Böhmová et al. (2013))	$O(n)^\dagger$ (Gupta et al. (1979), see also Section 5.6.3)	$O(mn^{m+1})$ (implied by NLS-m)	-
	new	$O(n)^\dagger$ (Section 5.4)	-	$O(mn^m)$ (implied by NLS-m)	strongly NP-complete (Section 5.8)
Bi-directional	notation	NLS-2	NLS-id	NLS-m	NLS
	known	-	$O(n \log n)$ (Felsner et al. (1997))	$O(mn^{m+1})$ (Sung and Vlach (2005))	-
	new	$O(n^2)$ (Section 5.5)	$O(n)^\dagger$ (Section 5.6.3)	$O(mn^m)$ (Section 5.7)	strongly NP-complete (implied by NLS-uni)

Table 5.1: Overview of notation for problem variants, known results, and results discussed in this chapter. † These results apply to input with arrival times given in sorted order, see Section 5.1.1.

Graph and concepts

An instance $\mathcal{I}$ is given by specifying distinct ordered arrival times $A_1 < \ldots < A_n$ and lockage durations $T_1 < T_2$. We say that an instance is feasible if there exists a no-wait solution, otherwise it is not feasible. Given an instance $\mathcal{I}$, we create a graph $G(\mathcal{I})$; we will, in the sequel, simply write G. Notice that we allow multiple edges between a pair of nodes in G; more precisely, the edge set of G consists of a set E^S and a set E^L. The graph is constructed as follows. There is a node for each arrival time; two nodes $i < j$ are connected via an *s-edge* $(i, j) \in E^S$ if and only if $A_j - A_i < 2T_1$; two nodes $i < j$ are connected via an *l-edge* $(i, j) \in E^L$ if and only if $A_j - A_i < 2T_2$. The following property underlies this construction: for a solution to be no-wait, two arrivals connected by an l-edge cannot both be served by the long chamber; two arrivals connected by an s-edge cannot both be served by the short chamber.

Observe that $E^S \subseteq E^L$, i.e. the existence of an s-edge (i, j) implies the existence of an l-edge (i, j), so that two arrivals connected by an s-edge can also not both be served by the long chamber. Further note that (V, E^S) and (V, E^L) are unit interval graphs by construction. In fact, deciding whether the set of nodes V can be partitioned into two subsets which are independent sets in E^S and E^L respectively, corresponds exactly to the problem stated in the "graph"-description of our problem, described in Section 5.2, for $m = 2$, where $E^S = E_1$ and $E^L = E_2$.

Because the arrival times are assumed to be strictly ordered, this same ordering applies to the nodes of V. Node i then refers to the arrival of a ship at time A_i, with $1 \leq i \leq n$. We may thus refer to a 'first' and 'last', or 'earlier and 'later' nodes without ambiguity. We call a pair of nodes (i, j) consecutive when $j = i + 1$. In all figures, we represent an s-edge by a straight line segment (—), while an l-edge is represented by a segment in the form of an arc (⌒).

A no-wait solution to the given instance can exist only if each ship can be assigned to one of the two lock chambers so that a lockage starts at the ship's time of arrival. We can observe a number of interesting properties in graph G. In addition to highlighting some structural characteristics of an instance and its graph, we will refer to these observations in the proof that follows.

Observation 5.1. If there exists a node $i \in V \setminus \{n\}$ such that $(i, i+1) \notin E^L$, i.e. if there exists a pair of consecutive nodes which are not connected by an l-edge, the instance splits into two independent

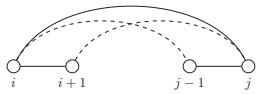


Figure 5.1: Structure described in Observation 5.3. Note that the existence of $(i, j) \in E^L$ implies that $(i, j-1) \in E^L$ and $(i+1, j) \in E^L$ since $A_i < A_{i+1} < A_{j-1} < A_j$ and since (V, E^L) is a unit interval graph.

sub-problems. This is easily seen since both chambers are then available at time A_{i+1} for any no-wait solution, regardless of the assignment of nodes $1, \ldots, i$. A no-wait solution thus exists if and only if there is a no-wait solution for each of the two sub-problems.

Observation 5.2. If there exists a node $i \in V$ such that $(i, i+2) \in E^S$, i.e. if there exists a triangle of s-edges in G, then the instance is not feasible. This readily follows from the fact that there does not exist a proper 2-colouring in a triangle graph.

Observation 5.3. If an l-edge contains two s-edges that are node-disjoint, i.e. if there exist nodes $i, j \in V$ with $j > i+2$ so that $(i, i+1) \in E^S$, $(j-1, j) \in E^S$, and $(i, j) \in E^L$, then the instance is not feasible. This is easily argued as follows: the existence of the s-edges implies that either node i or node $i+1$ must be served by the long chamber, and that either node $j-1$ or node j must be served by the long chamber. The l-edge (i, j), however, implies the existence of the l-edges $(i, j-1)$ and $(i+1, j)$, conflicting with the observation that two nodes must be served by the long chamber. An example is shown in Figure 5.1.

Observation 5.4. If there exists a node $i \in V$ such that there exist s-edges $(i, i+1), (i+1, i+2), (i+2, i+3) \in E^S$ and l-edges $(i, i+2), (i+1, i+3) \in E^L$, the instance is not feasible. Figure 5.2 shows the graph for an instance consisting of 4 such nodes. It is easily verified (see also the next observation) that both nodes $i+1$ and $i+2$ must be served by the short chamber in all feasible solutions, which is impossible due to the presence of s-edge $(i+1, i+2)$. Hence, an instance containing this structure is not feasible.

For convenience, we restrict ourselves in the remainder of Section 5.4 to instances $\mathcal{I}$ whose corresponding graphs $G(\mathcal{I})$ do not contain any of

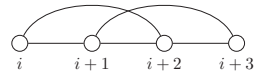
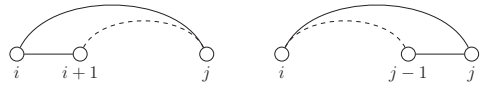


Figure 5.2: Structure described in Observation 5.4.



(a) (b)

Figure 5.3: Structures described in Observation 5.5. Note that the existence of the dashed edges is implied since (V, E^L) is an interval graph.

the structures described in Observations 5.1 to 5.4. This can be done without loss of generality, since we can recognize these structures efficiently, see Section 5.4.1. Furthermore, an additional observation allows us to recognize certain nodes which must be assigned to the short chamber:

Observation 5.5. If there exist nodes $i, j \in V$ with $j > i + 1$ such that $(i, i + 1) \in E^S$ and $(i, j) \in E^L$, a feasible solution can exist only when node j is assigned to the short chamber. This follows readily from the fact that either i or $i + 1$ must be assigned to the long chamber, and an l-edge to j starts at both i and $i + 1$. From symmetry, it immediately follows that the same holds for a node i where $(i, j) \in E^L$ and $(j, j - 1) \in E^S$. The associated graphs are shown in Figures 5.3a and 5.3b.

We proceed by describing paths and nodes in the graph which have a specific structure. We show later that checking for the presence of such paths suffices to decide on the feasibility of a given instance.

Definition 5.1. *Given an instance $\mathcal{I}$ and graph $G(\mathcal{I})$, a* bad path *is any sequence of distinct nodes $(i_1, i_2, \ldots, i_k)$ with $k \geq 4$ that satisfies:*

1. *The nodes in the sequence appear in the order defined on V, with exception of i_1 and i_k, which satisfy $i_2 < i_1 < i_3$ and $i_{k-2} < i_k < i_{k-1}$. More formally: $i_x < i_{x+1}$ for all $x \in \{2, \ldots, k-2\}$, $i_2 < i_1 < i_3$, and $i_{k-2} < i_k < i_{k-1}$.*

2. *The pairs of consecutive nodes in the sequence are alternatingly connected by an s-edge and an l-edge, with the first and last edges*

Figure 5.4: Example of a bad path. Note that the structure described in Definition 5.1 remains satisfied if the dashed edges are replaced with a longer sequence (with odd number of nodes) consisting alternatingly of s-edges and l-edges.

in the sequence being both s-edges. More formally: $(i_x, i_{x+1}) \in E^S$ *for all odd* $x \in \{1, \ldots, k-1\}$, $(i_x, i_{x+1}) \in E^L$ *for all even* $x \in \{1, \ldots, k-1\}$, *and* k *is even.*

Note that a bad path necessarily contains an even number of nodes. An example is shown in Figure 5.4. Observe that a bad path with $k = 4$ consists of the structure shown in Figure 5.1.

We also describe specific nodes that are closely related to the definition of a bad path. Intuitively, we define a *potentially bad node* as the latest node in a path $(i_1, i_2, \ldots, i_k)$ that contains at least 5 nodes, and that could be extended to a bad path if there would exist a node j with $i_{k-1} < j < i_k$ and an s-edge (j, i_k). Note that the existence of such a node j is not required for i_k to be a potentially bad node. More formally, we define a potentially bad node as follows.

Definition 5.2. *A node* i_k *is a* potentially bad node *if there exists a path* $(i_1, i_2, \ldots, i_k)$ *that satisfies the following:*

- $i_x < i_{x+1}$ *for all* $x \in \{2, \ldots, k-1\}$ *and* $i_2 < i_1$.
- $(i_x, i_{x+1}) \in E^S$ *for all odd* $x \in \{1, \ldots, k-1\}$ *and* $(i_x, i_{x+1}) \in E^L$ *for all even* $x \in \{1, \ldots, k-1\}$.
- $k \geq 5$ *and* k *is odd*

Observe that in a bad path, all nodes i_j in the path with $j \geq 5$ and j odd are potentially bad nodes. E.g. in Figure 5.4, nodes i_5 and i_7 are potentially bad nodes, whereas i_3 is not.

Characterizing feasible instances

We present the following theorem to characterize when an instance is feasible:

Theorem 5.1. *An instance $\mathcal{I}$ of NLS-uni-2-distinct is feasible if and only if its corresponding graph $G(\mathcal{I})$ does not contain a bad path.*

Proof. $\Rightarrow$ We argue that if $G(\mathcal{I})$ contains a bad path, the instance is not feasible. Consider a bad path $(i_1, \ldots, i_k)$ in G. It follows from Observation 5.5 and $i_2 < i_1 < i_3$ that node i_3 must be assigned to the short chamber. We now trace the path from i_3 to i_{k-1}. Note that, since l-edges and s-edges alternate, any solution must assign nodes $i_3, \ldots, i_{k-1}$ to the short and long chamber in alternating order. This implies that node i_{k-2} must be assigned to the long chamber, while node i_{k-1} must be assigned to the short chamber. However, the bad path implies that $(i_{k-1}, i_k) \in E^S$ and $(i_{k-2}, i_k) \in E^L$ so that no chamber is available for i_k and hence no feasible solution exists.

$\Leftarrow$ We now argue by contradiction that a feasible solution exists whenever the graph does not contain a bad path. For this, let us assume that there exists an instance which is not feasible while its corresponding graph does not contain a bad path. Assuming that such instances exist, consider one with a minimum number of arrivals. Let $G^* = (V^*, E^*)$ be the graph corresponding to this instance, with $E^* = E^{S*} \cup E^{L*}$ where E^{S*} and E^{L*} denote the set of s-edges and l-edges respectively. The proof is organized as follows. First, we show that G^* must contain at least one potentially bad node. We then argue that there cannot be a latest potentially bad node in G^*: a contradiction.

We claim that graph G^* must satisfy the following properties:

Property 5.1. *For each $i \in V^*$, there exists a feasible solution in $G^* \setminus \{i\}$ with distinct nodes $j_s, j_l \in V^* \setminus \{i\}$ such that $(i, j_s) \in E^{S*}$, $(i, j_l) \in E^{L*}$ while j_s (j_l) is assigned to the short (long) chamber in that feasible solution.*

Proof. Let i be an arbitrary node in G^*. Since G^* is a counterexample with a minimum number of nodes, and since no bad paths are introduced by removing a node and its incident edges, it follows immediately that a feasible assignment of chambers is possible in $G^* \setminus \{i\}$. Consider an arbitrary feasible assignment in $G^* \setminus \{i\}$ and let S and L be the set of nodes that are assigned to the short and long chamber in this solution respectively; clearly, $S \cap L = \emptyset$. If, in G^*, none of the nodes in S are connected to i by an s-edge, it is easily seen that node i can be assigned to the short chamber, thus extending the solution in $G^* \setminus \{i\}$ to a feasible solution in G^*. Since no feasible solution is possible in G^*, it follows that a node $j_s \in S$ must exist such that $(i, j_s) \in E^{S*}$. Repeating this reasoning

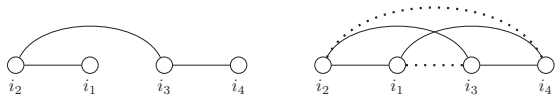


Figure 5.5: Structures described in case 1 in the proof. Note that, in (b), the dotted edges are not present in G^*, and a feasible solution assigns i_1 and i_3 to the short chamber and i_2 and i_4 to the long chamber.

for the long chamber immediately implies that there exists a node $j_l \in L$ such that $(i, j_l) \in E^{L*}$. □

Property 5.2. *G^* contains at least one potentially bad node.*

Proof. Consider the earliest node, say i_2, in G^*. Applying Property 5.1 to this node, it follows that there must exist two additional nodes i_1, i_3 such that $(i_2, i_1) \in E^{S*}$ and $(i_2, i_3) \in E^{L*}$. We select i_1, i_3 to be minimal, i.e. i_1 is the successor of i_2 and i_3 is the successor of i_1. We obtain the structure shown in Figure 5.3. Applying Property 5.1 to node i_3 implies that there is an s-edge incident to node i_3. We can distinguish two possible cases:

1. The s-edge is incident to a node $i_4 > i_3$. Observe that i_4 must the be the successor of i_3. We obtain the structure shown in Figure 5.5a. If G^* consists of four nodes, applying Property 5.1 to i_1 implies that $(i_1, i_4) \in E^{L*}$. Observe that i_3 cannot be selected as node j_l in the property, since i_3 must not be assigned to the long chamber (Observation 5.5). Furthermore, observe that $(i_1, i_3) \notin E^{S*}$ (by our assumption that the structure from Observation 5.4 is not present in an instance) and that $(i_2, i_4) \notin E^{L*}$ (by our assumption that the structure from Observation 5.3 is not present). Then, however, G^* corresponds to an instance that is feasible, which is not the case. This is illustrated in Figure 5.5b. Thus, G^* consists of at least five nodes. Observation 5.1 implies that $(i_4, i_5) \in E^{L*}$, and thus i_5 is a potentially bad node.

2. The s-edge is incident to node i_1. Observe that if G^* consists of only 3 nodes, a feasible solution is easily found. Thus, there exists a fourth node i_4 in G^*. Since i_2, i_1, and i_3 are the earliest nodes in the instance, i_4 is the successor of i_3. Applying Property 5.1 to

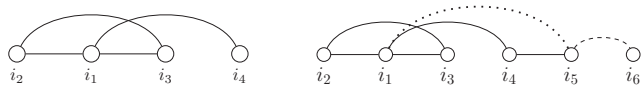


Figure 5.6: Structures described in case 2 in the proof.

node i_1, it follows that $(i_1, i_4) \in E^{L*}$. Observe that nodes i_2 and i_3 cannot be selected as node j_l in the property since they must not be assigned to the long chamber (Observation 5.5). G^* then contains the structure shown in Figure 5.6a. We then apply Property 5.1 to node i_4. Since we may assume that $(i_3, i_4) \notin E^{S*}$ (otherwise the case described above applies), it follows that there exists a node $i_5 > i_4$ with $(i_4, i_5) \in E^{S*}$. The resulting structure is shown in Figure 5.6b. Finally, we apply Property 5.1 again, now to i_5. It follows that there is an l-edge incident to i_5. As before, note that nodes i_3 and i_4 cannot be chosen as node j_l in the property since they must not be assigned to the long chamber (Observation 5.5). Since the existence of the l-edge (i_1, i_5), dotted in Figure 5.6b, implies a bad path, this l-edge cannot be present and there thus exists a node $i_6 > i_5$ with $(i_5, i_6) \in E^{L*}$. Node i_6 is then a potentially bad node, with path $(i_3, i_1, i_4, i_5, i_6)$ satisfying the definition above.

Thus, G^* must contain at least one potentially bad node. □

Property 5.2 implies that G^* contains a latest potentially bad node. We now show that this leads to a contradiction. Figure 5.7 illustrates the reasoning below. Let i_k be the latest potentially bad node in G^*. Applying Property 5.1 to i_k implies that there exists an s-edge (i_k, i_{k+1}). Since i_k is a potentially bad node, $i_{k+1} < i_k$ would imply the existence of a bad path. Node i_{k+1} is thus the successor of i_k. Applying Property 5.1 again, now to i_{k+1}, implies the existence of an additional l-edge. Observe that the l-edge (i_{k-1}, i_{k+1}), illustrated by the dotted l-edge in Figure 5.7, cannot be present because it implies the existence of a bad path. Thus, there exists a node $i_{k+2} > i_{k+1}$ with $(i_{k+1}, i_{k+2}) \in E^{L*}$. Note that i_{k+2} is a potentially bad node and $i_{k+2} > i_k$. This contradicts our choice of i_k as the latest potentially bad node.

Thus, the existence of G^* leads to a contradiction, proving the theorem. □

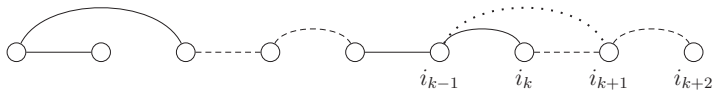


Figure 5.7: Structure describing the latest potentially bad node in G. Observe that the dotted l-edge cannot be present.

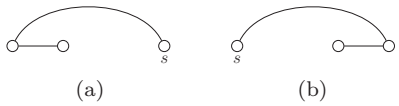


Figure 5.8: The nodes labeled s must be assigned to the short chamber.

An $O(n)$ algorithm for deciding feasibility and constructing a solution

Notice that Theorem 5.1 characterizes feasibility of an instance. We now present an $O(n)$ algorithm that actually recognizes whether G contains a bad path. We start with the arrival times A_i for $i \in \{1, \ldots, n\}$ in sorted order, and avoid explicitly constructing G, which may contain up to $O(n^2)$ l-edges. Since (V, E^L) is a unit interval graph, we can avoid checking each of the l-edges explicitly. Instead of constructing all edges in graph G, we will check for the existence of edges, using their definition, only when needed. For example, when we write "if $(v_i, v_j) \in E^L$" for nodes $v_i, v_j \in V$ in what follows, this equals the expression "if $A_j - A_i < 2T_2$", corresponding to the definition of an l-edge. For any given pair of nodes, this is easily checked in constant time. In addition to recognizing feasibility, the algorithm assigns each node to a chamber such that the corresponding solution is a no-wait solution, provided that no bad path exists.

Recall that in Figure 5.8a and Figure 5.8b, each of the nodes labelled s must be assigned to the short chamber in any feasible solution, as argued in Observation 5.5.

The outline of the procedure is as follows. We first identify all nodes that must be assigned to the short chamber due to the structures shown in Figure 5.8a and Figure 5.8b. We then use implications from these assignments to assign other nodes to the chambers. In this way, all 'forced' assignments are handled. Finally, we apply a simple greedy procedure to assign the remaining nodes to chambers. In the remainder of this section, we will show that if no bad paths are present, the greedy procedure always

yields a feasible assignment of lock chambers (i.e. the correctness of the algorithm), and that each of these steps can be performed in linear time (i.e. the complexity of the algorithm).

We will call any node that must be assigned to the same chamber in all feasible solutions due to the implications mentioned above, a *fixed node*. We distinguish *s-fixed* and *l-fixed* nodes for nodes that must be assigned to the short and long chambers respectively. As argued in Observation 5.5, the nodes labelled s in Figure 5.8 are s-fixed nodes. We start by identifying all occurrences of these structures in G. Observe that these structures correspond to the 'beginning' (i.e. the first three nodes) and the 'end' (i.e. the last three nodes) of a bad path. Finding these structures is easily done in linear time by considering each node once and checking for the presence of the edges shown in the figures. We obtain an initial set of s-fixed nodes. Initially, no nodes are l-fixed.

Observe that any node connected to an s-fixed node by an s-edge is necessarily l-fixed and that any node connected to an l-fixed node by an l-edge is necessarily s-fixed. The following step is to identify all remaining nodes that can be fixed using these observations. We first consider all nodes in order; let i be the current node. If i is s-fixed and $(i, i+1) \in E^S$, add $i+1$ to the set of l-fixed nodes. If i is l-fixed, add all $j > i$ for which $(i, j) \in E^L$ to the set of s-fixed nodes. Next, we repeat this for all nodes in reverse order; again let i be the current node. If i is s-fixed and $(i, i-1) \in E^S$, add $i-1$ to the set of l-fixed nodes. If i is l-fixed, add all $j < i$ for which $(i, j) \in E^L$ to the set of s-fixed nodes. Clearly, when a node is both s-fixed and l-fixed, no feasible solution exists. Observe that such a conflict is only possible if a path starting from the 'beginning' of a bad path is extended to a node where it meets the 'end' of a bad path.

It is important to note that nodes fixed by considering all nodes in order do not in turn fix any earlier nodes. Similarly, nodes fixed by considering all nodes in reverse order have no impact on later nodes. This is illustrated in Figure 5.9. In Figure 5.9a, assume that i is an s-fixed node. Adding $i+1$ to the set of l-fixed nodes then has no new implications for any nodes earlier than i because if an l-edge $(j, i+1)$ exists with $j < i$, the initial set of s-fixed nodes already contained node j. In the rightmost figure, assume that i is an l-fixed node. Adding $i+1$ to the set of s-fixed nodes then has no implications for any nodes earlier than i since the existence of an s-edge $(j, i+1)$ with $j < i$ implies a triangle of s-edges, which is not present due to Observation 5.2. The similar statement where nodes are considered in reverse order follows immediately from symmetry.

It follows that after considering all nodes once in order and once in

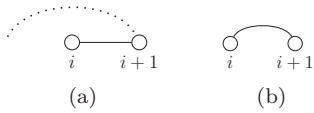


Figure 5.9: Assigning $i + 1$ to a chamber has no implications for nodes earlier than i.

reverse order, no new nodes can be fixed. As noted above, if a bad path exists, at least one node must be both s-fixed and l-fixed. If no such contradiction is found, the instance is thus feasible. In the corresponding solution, all s-fixed nodes are assigned to the short chamber and all l-fixed nodes are assigned to the long chamber. Upon completing this step, there may remain nodes with no chamber assignment. Note that none of the fixed nodes has any implications for any of the remaining nodes, since this would imply that additional nodes should be fixed nodes. We describe a straightforward greedy rule that assigns the remaining nodes to the chambers and show that applying this rule to a node does not prevent us from applying it to any later node, thus yielding a feasible solution. The greedy procedure considers all nodes in order. Let i be the current node to be labelled: if $(i-1, i) \in E^S$ and $i-1$ is assigned to the short chamber, assign i to the long chamber; otherwise assign i to the short chamber. To see that applying this rule does not prevent us from applying it to any later node, consider node i in Figure 5.10. If $(i, i+1) \in E^S$, then node $i+1$ must be assigned to the long chamber only if i is assigned to the short chamber; this is consistent with the rule. If there exists a $j > i+1$ for which $(i, j) \in E^L$, no s-edges may be incident to any nodes k with $i < k < j$ because this would imply that either i or j is a fixed node and hence is already assigned to a chamber. All such nodes k, as well as node j can thus be assigned to the short chamber, which is consistent with the rule. Note that it cannot be the case that both $(i, i+1) \in E^S$ and there exists a $j > i+1$ with $(i, j) \in E^L$ since this would imply that j is a fixed node. After applying the greedy rule, all nodes have been assigned to a chamber; if no bad path was identified, we have thus obtained a feasible solution. A pseudo-code of this procedure is presented in Algorithm 3. Note that the algorithm assumes that $n \geq 3$; instances with $n < 3$ can be solved trivially. The discussion above establishes correctness of the algorithm.

We remark that the greedy rule described above is applied only after

```
input: arrival times t_1 < t_2 < ... < t_n, lockage durations T_1 < T_2
v_i ← node corresponding to the arrival at time A_i, for all i ∈ {1, ..., n}
s-fixed ← ∅,    l-fixed ← ∅
// identify initial s-fixed nodes:
j = 1
for i = 1 to |V| do
    j ← max(i + 2, j)
    if (v_i, v_{i+1}) ∈ E^S then
        while (v_i, v_j) ∈ E^L do
            s-fixed ← s-fixed ∪ {v_j}
            j ← j + 1
j = |V|
for i = |V| to 1 do
    j ← min(i − 2, j)
    if (v_{i−1}, v_i) ∈ E^S then
        while (v_j, v_i) ∈ E^L do
            s-fixed ← s-fixed ∪ {v_j}
            j ← j − 1
// extend implications of fixed nodes:
for i = 1 to |V| do
    if v_i ∈ s-fixed and (v_i, v_{i+1}) ∈ E^S then l-fixed ← l-fixed ∪ {v_{i+1}}
    j ← i + 2
    if v_i ∈ l-fixed then
        while (v_i, v_j) ∈ E^L do
            s-fixed ← s-fixed ∪ {v_j}
            j ← j + 1
for i = |V| to 1 do
    if v_i ∈ s-fixed and (v_{i−1}, v_i) ∈ E^S then l-fixed ← l-fixed ∪ {v_{i−1}}
    j ← i − 2
    if v_i ∈ l-fixed then
        while (v_j, v_i) ∈ E^L do
            s-fixed ← s-fixed ∪ {v_j}
            j ← j − 1
if s-fixed ∩ l-fixed ≠ ∅ then return 'not feasible'
// assign nodes to chambers using a greedy rule:
for i = 1 to |V| do
    if v_i ∈ s-fixed then chambers_i ← 'short'
    else if v_i ∈ l-fixed then chambers_i ← 'long'
    else if (v_{i−1}, v_i) ∈ E^S and chambers_{i−1} = 'short' then
        chambers_i ← 'long'
    else chambers_i ← 'short'
```

Algorithm 3: Pseudo-code algorithm for problem SLS-uni-2-distinct.

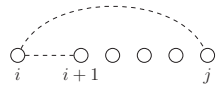


Figure 5.10: Illustration of the assignment rule for remaining nodes. Note that only one of the dashed edges may be present for any $j > i + 1$.

Figure 5.11: Example showing that the greedy rule does not yield a feasible solution in general. Note that node i_2 can be identified as an s-fixed node.

no additional nodes can be fixed. Figure 5.11 illustrates that, in contrast to the above, applying the rule may not yield a feasible solution if not all fixed nodes have been identified. In the figure, one feasible solution consists of assigning nodes i_2 and i_3 to the short chamber and nodes i_1 and i_4 to the long chamber. The greedy rule, however, assigns node i_1 to the short chamber and node i_2 to the long chamber, so that no feasible assignment remains for nodes i_3 and i_4.

It remains to show that the procedure described above runs in linear time. As argued above, finding the initial set of s-fixed nodes takes at most $O(n)$ time. Finding all additional s-fixed nodes is also possible in linear time. Indeed, each node has at most two s-edges incident to it and each node is considered only once in order, and once in reverse order. Graph G contains up to $O(n^2)$ l-edges; to see that finding all additional l-fixed nodes takes only $O(n)$ time, we make use of the fact that (V, E^L) is an interval graph. By definition, it follows that if l-edge (u, v) exists, all l-edges (u, w) with $u < w < v$ must also exist. Thus, when traversing the nodes in order, if a node was labelled as s-fixed due to the presence of an l-edge, no nodes earlier than this fixed node need to be checked at a later time, since such nodes are already s-fixed. Thus, it suffices to remember the latest node that was s-fixed to avoid checking all possible l-edges for each of the nodes. In Algorithm 3, this is achieved by keeping track of j, which represents the next node to be checked for the existence of an l-edge. The same argument applies when considering nodes in reverse order, where it suffices to start from the earliest s-fixed node. Since in each iteration either i or j increases and no nodes earlier than j are checked, finding the

s-fixed nodes completes in $O(n)$ time.

Finally, we note that recognizing the structures described in Observations 5.1 to 5.4, which were assumed not to be present in the graph, can be achieved in linear time. This is easily seen for Observations 5.1, 5.2 and 5.4, which deal only with adjacent nodes. To recognize these structures, it suffices to traverse each of the nodes and check for the existence of the incident edges described in the observations. For Observation 5.3, we use the interval graph structure in the same was as when recognizing the s-fixed nodes in Algorithm 3. That is, we remember the latest node that was s-fixed in order to avoid checking earlier nodes which are already known to be s-fixed. If any of these structures is identified, it immediately follows that no feasible solution exists.

5.4.2 Fixed chamber assignments

We describe here how the algorithm can be extended to the case where some nodes are pre-assigned to a specific chamber. Note that when specific chamber assignments are imposed, an instance may not have a feasible solution while its corresponding graph does not contain a bad path. Our algorithm, however, is able to take these given assignments into account.

If a given subset of the nodes, say $S \subseteq V$, is pre-assigned to the short chamber, we initialize, in Algorithm 3, the set of s-fixed nodes to this set S. Similarly, if a given subset of nodes, say $L \subseteq V$, is pre-assigned to the long chamber, we initialize, in Algorithm 3, the set of l-fixed nodes to this set L. Clearly, when a node is both s-fixed and l-fixed, the instance is not feasible. Once the sets of fixed nodes are initialized, the initial sets of fixed nodes are extended as in Algorithm 3. The analysis of the assignment rule for all nodes that are not fixed remains valid. Note that feasibility is still identified by verifying for each node whether it is both s-fixed and l-fixed. These adjustments suffice to solve the generalization with fixed chamber assignments. The computational complexity remains unchanged.

5.4.3 Simultaneous arrivals

In this section, we will consider the generalization where simultaneous arrivals can occur, i.e. where arrival times need not be distinct. Since a chamber may then simultaneously serve more than one ship in a no-wait solution, we need to take the capacity of the chambers into account. Let $C_{\text{small}} = \min(C_1, C_2)$ and $C_{\text{large}} = \max(C_1, C_2)$. Note that C_{small} does not necessarily correspond to the chamber with the shortest lockage

duration. We now show how to modify graph G and Algorithm 3 in order to determine whether a feasible solution exists in this setting with simultaneous arrivals.

For each arrival time $t \in \mathcal{A}$, let k_t be the number of ships arriving at time t. Clearly, if there exists such a t with $k_t > C_{\text{small}} + C_{\text{large}}$, the instance is not feasible since the number of arriving ships at time t exceeds the combined capacity of both chambers. Let us thus assume that the instance has $k_t \leq C_{\text{small}} + C_{\text{large}}$ for all $t \in \mathcal{A}$. For each $t \in \mathcal{A}$, we distinguish three cases:

1. Case 1: $2 \leq k_t \leq C_{\text{small}}$. We modify graph G as follows: we let a single node represent all the simultaneous arrivals at time t. Either chamber is a valid assignment to simultaneously serve all k_t ships; we may thus treat these simultaneous arrivals as a single ship.

2. Case 2: $C_{\text{small}} < k_t \leq C_{\text{large}}$. As in the case above, let a single node represent all simultaneous arrivals at time t. In addition, we impose that this node must be assigned to the large chamber. Recall that the large chamber may be either the short or the long chamber, depending on the values of T_1, T_2, C_1, C_2. It is easily seen that it is never required to use both chambers to serve these k_t ships, since the large chamber must be used regardless, and this chamber suffices to serve all ships simultaneously.

3. Case 3: $C_{\text{large}} < k_t \leq C_{\text{small}} + C_{\text{large}}$. Again, let a single node represent all simultaneous arrivals at time t. It follows that since $k_t > C_{\text{large}}$, both chambers must be used simultaneously to transfer all ships without introducing waiting time. We add this node to the set B designated to identify all nodes that must be assigned to both chambers. We argue below that, after modifying the graph for all $t \in \mathcal{A}$, the algorithm is easily adjusted to enforce this assignment for each node in B.

After graph G has been modified, we take B into account by initializing the algorithm as follows. For each node $i \in B$, we add all implications that follow from imposing that i is assigned to both chambers: for all $j \in V$ with $(i, j) \in E^S$, add j to the set l-fixed; for all $j \in V$ with $(i, j) \in E^L$, add j to the set s-fixed. It is easily argued that each of the added implications must hold in any feasible solution. When assigning the nodes to chambers, we set $\text{chambers}_i =$ 'short + long' for all $i \in B$. Furthermore, for a feasible solution to exist, it must hold that none of the nodes in B are fixed

when running the algorithm. Indeed, if a node $k \in B$ with corresponding arrival time A_k would be fixed, this implies that at least one chamber is unavailable at time A_k, so that there is no feasible assignment for k. In addition to modifying the initialization of sets s-fixed and l-fixed, we thus extend the check for conflicts in the algorithm to "if s-fixed $\cap$ l-fixed $\neq \emptyset$ or s-fixed $\cap$ $B \neq \emptyset$ or l-fixed $\cap$ $B \neq \emptyset$".

Using the result for fixed chamber assignments described in Section 5.4.2, and by modifying Algorithm 3 as outlined above, it follows that we can solve the resulting instance in $O(n)$ time. The following theorem concludes this discussion.

Theorem 5.2. *If a no-wait solution exists for the uni-directional lock scheduling problem with two chambers it can be found, i.e. problem NLS-uni-2 can be solved, in $O(n)$ time.*

5.5 Two arbitrary chambers

We now focus on the more general setting with two lock chambers, where ships may travel in both directions. Böhmová et al. (2013) mention a reduction to 2-SAT for an interval scheduling problem which generalizes the uni-directional setting. The well-known 2-SAT problem can be solved in a number of operations which is linear in the number of clauses, see Even et al. (1976) and Aspvall et al. (1979). We show here that the same applies to the bi-directional two-chamber setting and describe the reduction explicitly.

Lemma 5.1. *An instance of NLS-2 can be modelled as an instance of 2-SAT using $O(n)$ variables and $O(n^2)$ clauses.*

Proof. In the NLS-2 setting ships may arrive simultaneously. To take this into account, we first describe how each instance of NLS-2 can be transformed into an equivalent instance where $C_1 = C_2 = 1$ and where some ships are pre-assigned to chambers. We then provide a reduction to 2-SAT for the setting with two unit-capacity chambers and pre-assigned ships.

Similar to the approach followed in Section 5.4.3, we distinguish multiple cases when constructing the instance with unit capacity. For each time $t \in \mathcal{A}$, let $k_{t,d}$ be the number of ships arriving at time t and travelling in direction d. Clearly, the instance is not feasible if there exist a t and d such that $k_{t,d} > C_1 + C_2$. Since this can be easily verified, we assume that $k_{t,d} \leq C_1 + C_2$ for all $t \in \mathcal{A}$, $d \in \{\text{upstream, downstream}\}$.

Consider a given instance $\mathcal{I}$ of NLS-2. As in Section 5.4.3, let $C_{\text{small}} = \min(C_1, C_2)$ and $C_{\text{large}} = \max(C_1, C_2)$. We construct an instance $\mathcal{I}_{\text{unit}}$, where $C_1^{\mathcal{I}_{\text{unit}}} = C_2^{\mathcal{I}_{\text{unit}}} = 1$, leaving the lockage times and the set of arrival times unaltered. The set of arriving ships is constructed as follows. For each $t \in \mathcal{A}$ and $d \in \{\text{upstream, downstream}\}$:

1. If $k_{t,d} \leq C_{\text{small}}$, replace these $k_{t,d}$ ships by a single ship travelling in direction d, arriving at time t. It is immediately clear that either lock chamber suffices to handle all $k_{d,t}$ ships, so that we effectively ignore these simultaneous arrivals.

2. If $C_{\text{small}} < k_{t,d} \leq C_{\text{large}}$, replace all these ships by a single ship travelling in direction d, arriving at time t, and additionally impose that this ship must be assigned to the large chamber (i.e. the second chamber if $C_1 \leq C_2$, the first chamber otherwise). To serve all ships without introducing waiting time, the large chamber must be used to serve at least one ship arriving at time t and travelling in direction d. By pre-assigning the ship to the large chamber, it follows that this chamber must be available at time t to serve ships travelling in direction d. In fact, the capacity of the large chamber suffices to serve all ships arriving at time t and travelling in direction d. Consequently, we can ignore the simultaneously arriving ships in what follows.

3. If $C_{\text{large}} < k_{t,d}$, replace all these ships by two ships travelling in direction d, arriving at time t. Further, pre-assign the first ship to the first chamber and the second ship to the second chamber. It follows that both chambers must be available at time t to serve ships travelling in direction d.

The resulting instance is called $\mathcal{I}_{\text{unit}}$. Finding a no-wait schedule for $\mathcal{I}_{\text{unit}}$ with pre-assigned ships thus yields a no-wait solution for the original instance $\mathcal{I}$ of NLS-2. We now describe a reduction to 2-SAT for the problem of finding a no-wait solution for instances with two unit-capacity chambers and pre-assigned ships.

Recall that a no-wait schedule exists if and only if each ship can be assigned to either the short or the long chamber such that, for each chamber, lockages do not overlap. For lock chamber $c \in \{1, 2\}$, it is easily seen that there is no overlap if and only if $|A_s - A_{s'}| \geq T_c$ for each pair of ships s and s' that are assigned to chamber c, and in addition $|A_s - A_{s'}| \geq 2T_c$ if ships s and s' travel in the same direction, as argued in Section 5.1. We

create the following instance of 2-SAT: for each ship $s \in \mathcal{S}$, define a literal x_s. We will argue later that $x_s = \mathit{false}$ corresponds to assigning ship s to the short chamber, and $x_s = \mathit{true}$ corresponds to assigning ship s to the long chamber. Let the Boolean expression in conjunctive normal form consist of clauses described as follows. For each pair of ships s, $s' \in \mathcal{S}$:

1. if the ships travel in opposite direction and $|A_s - A_{s'}| < T_1$, add the clause $(x_s \vee x_{s'})$,

2. if the ships travel in the same direction and $|A_s - t_{s'}| < 2T_1$, add the clause $(x_s \vee x_{s'})$,

3. if the ships travel in opposite direction and $|A_s - A_{s'}| < T_2$, add the clause $(\neg x_s \vee \neg x_{s'})$,

4. if the ships travel in the same direction and $|A_s - A_{s'}| < 2T_2$, add the clause $(\neg x_s \vee \neg x_{s'})$.

Observe that, if $x_s = \mathit{false}$ where ship s is assigned to the short chamber and $x_s = \mathit{true}$ where it is assigned to the long chamber, $(\neg x_s \vee \neg x_{s'})$ suffices to prevent overlapping lockages for the long chamber whereas $(x_s \vee x_{s'})$ ensures that there is no overlap for the short chamber.

In addition, to enforce that all pre-assignments are respected, we add a clause containing only the literal x_s for all ships s that are pre-assigned to the long chamber, and a clause consisting of $\neg x_s$ for all ships s that are pre-assigned to the short chamber.

We claim that the existence of a truth assignment satisfying the Boolean formula is equivalent to the existence of a no-wait schedule. Indeed, given a truth assignment, we assign ship s to the short chamber if $x_s = \mathit{false}$ and to the long chamber if $x_s = \mathit{true}$. The definition of the clauses implies that no overlapping lockages exist for either chamber while each ship is assigned to a chamber, and hence we found a no-wait schedule. Also, the existence of a no-wait schedule immediately translates into a satisfying truth assignment. □

Note that the number of clauses in the 2-SAT instance described above, as well as the time needed to construct this instance, is quadratic in the number of ships. To find a no-wait solution for the NLS-2 problem, it follows that we can construct an instance of 2-SAT as described above, and use any algorithm for 2-SAT with a running time linear in the number of clauses. We can summarize this in the following theorem:

Theorem 5.3. *The bi-directional case for two arbitrary chambers, i.e. problem NLS-2, can be solved in $O(n^2)$ time.*

5.6 Identical chambers

We now focus on problem NLS-id, i.e. $C_c = C$ and $T_c = T$ for each $c \in \mathcal{C}$. Again we assume that arrival times are given in sorted order (Section 5.1.1). In fact, we consider a more general optimization version of NLS-id, where we aim at finding the minimum number of chambers allowing a no-wait solution. We first show that this problem is a special case of colouring trapezoid graphs. In Section 5.6.2, we provide a description of a greedy procedure and argue that it always finds a no-wait schedule while using a minimum number of chambers. We then prove in Section 5.6.3 that this procedure can be implemented with an $O(n)$ running time for both the uni-directional as well as the bi-directional case.

5.6.1 Colouring trapezoid graphs

For the definition of trapezoid graphs we consider a pair of parallel lines, labelled *up* and *down*. A trapezoid between these lines is defined by two points per line. Let this construction of lines and trapezoids be called the trapezoid instance. A graph $G = (V, E)$ is called a trapezoid graph if there exists a trapezoid instance with $|V|$ trapezoids, each corresponding to a node in V, such that there is an edge in E connecting nodes i and j if and only if the trapezoids corresponding to i and j intersect. See Figures 5.12 and 5.13 for an example illustrating this definition. Felsner et al. (1997) discuss the colouring of trapezoid graphs and show that a proper colouring with minimum number of colours can be found in $O(n \log n)$ time.

The special case of this trapezoid graph colouring problem that we consider is the following: given a trapezoid instance where all trapezoids are identical isosceles triangles, find a proper colouring with a minimum number of colours. We denote this problem by TC. Notice that while the triangles are identical, their orientation may differ depending on which of the parallel lines contains a single point. In a trapezoid colouring instance, we say that a triangle is *up-oriented* if this triangle has a single point on the up line and two points on the down line; a triangle is *down-oriented* if it has a single point on the down line and two points on the up line.

We argue that we can reduce NLS-id to TC, and vice versa. As in Section 5.5, let $k_{t,d}$ denote the number of ships arriving at time t and

travelling in direction d; we first argue that we can deal with simultaneous arrivals by transforming each instance $\mathcal{I}$ of NLS-id into an equivalent instance $\mathcal{I}_{\text{unit}}$ with unit capacity. In the remainder of Section 5.6, we can then restrict ourselves to instances where $C = 1$. Given $\mathcal{I}$, we construct $\mathcal{I}_{\text{unit}}$ as follows. For each $t \in \mathcal{A}$, replace the $k_{t,d}$ arrivals by $\lceil k_{t,d}/C \rceil$ arrivals at time t and travelling in direction d. Clearly, in $\mathcal{I}$, at least $\lceil k_{t,d}/C \rceil$ chambers are needed to serve these $k_{t,d}$ ships. It is also clear that any remaining capacity in the chosen lock chambers cannot be used to serve other ships without introducing waiting time. Consequently, a solution in $\mathcal{I}_{\text{unit}}$ corresponds directly to a solution in $\mathcal{I}$. Observe that constructing $\mathcal{I}_{\text{unit}}$ can be done in $O(n)$ time.

We thus turn our attention to the case with unit capacity. Given an instance of NLS-id where $C = 1$, we specify a set of identical isosceles triangles between parallel lines *up* and *down* as follows. For each downstream travelling ship s we construct a triangle with a point on the up line at t_s and points on the down line at $A_s - T$ and $A_s + T$; for each upstream travelling ship s we construct a triangle with a point on the down line at A_s and points on the up line at $A_s - T$ and $A_s + T$. From this construction, we can derive two fundamental properties:

1. The triangles corresponding to two ships s and s' travelling in the same direction intersect if and only if $|A_s - A_{s'}| < 2T$.

2. The triangles corresponding to two ships s and s' travelling in opposite directions intersect if and only if $|A_s - A_{s'}| < T$.

Note that in either case, ships s and s' cannot be served by the same chamber. Concluding, ships can be assigned to the same chamber if and only if the corresponding triangles do not intersect. Hence, a proper colouring of the corresponding graph represents a no-wait schedule where a colour refers to a chamber. The chromatic number of the trapezoid graph, then, represents the minimum number of chambers allowing a no wait schedule. Each instance of NLS-id can thus be modelled as an instance of TC. Similarly, it is easily seen that each instance of TC can be modelled as an instance of NLS-id.

In order to illustrate this reduction, we provide an example instance in Figure 5.12. We have downstream-travelling ships 1, 2, 4, and 6, arriving at times 10, 25, 42, and 54, and upstream-travelling ships 3 and 5, arriving at times 30 and 50. The lockage duration equals $T = 10$. Consequently, the pairs of ships that cannot both be served by a single chamber are (1,2), (2,3), (2,4), (4,5), (4,6), and (5,6). This is represented by intersections of

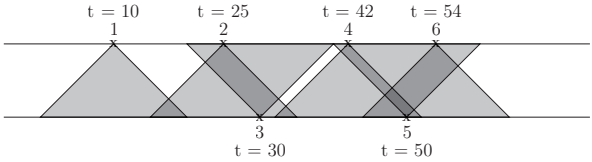


Figure 5.12: Example instance illustrating the definition of the corresponding trapezoids.

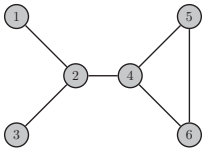


Figure 5.13: Trapezoid graph corresponding to the example instance of Figure 5.12.

triangles accurately. The graph corresponding to this instance is shown in Figure 5.13. It is immediately seen that at least three colours are required to colour the graph since it contains a clique on nodes 4, 5, and 6. One feasible solution could consist of assigning ships 1 and 4 to the first chamber, ships 2 and 5 to the second chamber, and ships 3 and 6 to the third chamber.

Since the chromatic number of a trapezoid graph can be found in $O(n \log n)$ time, it immediately follows that an $O(n \log n)$ algorithm exists that solves NLS-id. In the remainder of this section, we improve this result to yield an $O(n)$ algorithm.

5.6.2 Correctness of a greedy procedure for NLS-id

We argued in Section 5.6.1 that we can reduce each instance of NLS-id to an equivalent instance with $C = 1$. We thus restrict ourselves to the setting with $C = 1$.

We say that a chamber is available at time t for direction d if the last ship served by this chamber (say ship s) travels in direction d and has an arrival time $A_s \leq t - 2T$, or travels in the direction opposite to d and has an arrival time $A_s \leq t - T$. In the former case, we say that the availability period of the chamber equals $t - A_s - 2T$; in the latter case, the availability period of the chamber equals $t - A_s - T$. The

solution procedure is as follows. Initially, let the number of chambers be zero. Consider all arrival ships in order, let A_s and d_s be the arrival time and direction of ship s being considered. If no chambers are available at time A_s for direction d_s, add an additional chamber and assign ship s to this chamber; otherwise, assign ship s to the chamber with the largest availability period for direction d_s at time A_s.

We argue that this greedy procedure yields a solution with a minimum number of chambers. Whenever a chamber is added to serve ship s, there is no possible assignment of ships $1, \ldots, s-1$ to chambers so that any of the chambers is available at time A_s. Indeed, since all chambers are identical, the choice of which preceding ship is assigned to which chamber is in fact irrelevant. Thus, after considering a ship s, a chamber is added if and only if the current number of chambers is not sufficient to serve ships $1, \ldots, s$ without waiting time. It follows that after considering all ships, we have a no-wait solution which uses a minimum number of lock chambers.

5.6.3 An $O(n)$ algorithm for NLS-id

To see that the procedure from Section 5.6.2 can be implemented to run in linear time, we first briefly discuss the uni-directional setting before extending the implementation to the bi-directional case. When all ships travel in the upstream direction, it is easily seen that there is an optimal solution where a chamber, after transferring a ship, immediately returns to the downstream side. A chamber that serves a ship is then always unavailable for $2T$ time units, starting from the arrival time of that ship. Solving this problem corresponds to a basic interval scheduling problem, for which Ford and Fulkerson (1962) describe a 'staircase rule' based on Dilworth's chain decomposition theorem. Gupta et al. (1979) provide a more efficient algorithm, which runs in $O(n \log n)$ in general. When the intervals have equal length and are sorted by starting time, it is easily seen that an initial sorting step in this algorithm can be omitted, reducing the complexity to $O(n)$. Applying the simplified version of this algorithm thus immediately yields a solution to NLS-uni-id.

While this approach is straightforward for NLS-uni-id, the time at which a chamber becomes available depends on the direction of travel in the more general NLS-id. Indeed, a chamber that finishes an upwards lockage is immediately available to serve a downstream travelling ship; the next upstream travelling ship, however, can only be served after an additional T time units needed to return to the downstream side. As a

result, for a given time t and direction d, a chamber which started a lock movement at time t_1 may not be available while a different chamber which started a lock movement at time $t_2 > t_1$ is available.

The main challenge of implementing our greedy rule is thus to efficiently keep track of the different chambers, and the moments in time at which they are available to serve ships depending on their direction. To achieve this, we maintain the following lists throughout the solution procedure. Each of the entries that will be added to these lists consists of a pair (t, s), where t specifies the time at which the chamber that serves ship s becomes available for a given direction.

1. list $\mathcal{A}^{UU}$: availability to serve an upstream-travelling ship, when the last served ship s is upstream-travelling.
2. list $\mathcal{A}^{UD}$: availability to serve an upstream-travelling ship, when the last served ship s is downstream-travelling.
3. list $\mathcal{A}^{DU}$: availability to serve a downstream-travelling ship, when the last served ship s is upstream-travelling.
4. list $\mathcal{A}^{DD}$: availability to serve a downstream-travelling ship, when the last served ship s is downstream-travelling.

As outlined in the description in Section 5.6.2, we keep track of the required number of chambers m. Additionally, we follow up whether each chamber remains available as the algorithm runs. Throughout the algorithm, R_s are Boolean values indicating whether, after serving ship s, a chamber has been used to serve another ship ($1 \leq s \leq n$).

A pseudo-code for the algorithm is provided in Algorithm 4. In words: we consider each ship in order and first verify whether one of the existing chambers is available at the position where the ship arrives. Let s be the ship under consideration. If a chamber c is available, it contains an entry in two of the lists. Upon assigning a ship to c we update R_s so that the second entry corresponding to c becomes invalid. We update the times at which c becomes available for each direction. If no chamber is available, we update m and proceed as above.

To see that Algorithm 4 runs in linear time, note the following. Each ship is considered only once, in the given input order. New entries in the lists $\mathcal{A}^{UU}$, $\mathcal{A}^{UD}$, $\mathcal{A}^{DU}$, and $\mathcal{A}^{DD}$ are always added to the end of the list. Furthermore, within each list, the time value of newly inserted entries is non-decreasing with s since the increment when constructing the entry is the same for all entries within each of the lists; for example, all entries

```
input: arrival times A_1 ≤ A_2 ≤ ... < A_n, directions d_1, d_2, ..., d_n,
lockage duration T
A^UU ← ∅, A^UD ← ∅, A^DU ← ∅, A^DD ← ∅
R_s ← false, for all s ∈ S
m ← 0
for s = 1 to n do
    reUsed = false
    if d_s == downstream then
        (t*, s*) ← earliest entry in A^DU ∪ A^DD
        while t* ≤ t_s and reUsed == false do
            if R_s == false then
                reUsed = true
                R_s* ← true
            delete entry (t*, s*) from the list in which it is contained
            (t*, s*) ← earliest entry in A^DU ∪ A^DD
    else
        (t*, s*) ← earliest entry in A^UU ∪ A^UD
        while t* ≤ t_s and reUsed == false do
            if R_s == false then
                reUsed = true
                R_s* ← true
            delete entry (t*, s*) from the list in which it is contained
            (t*, s*) ← earliest entry in A^UU ∪ A^UD
    if reUsed == false then
        m ← m + 1
    if d_s == downstream then
        add entry (A_s + T, s) to the back of list A^UD
        add entry (A_s + 2T, s) to the back of list A^DD
    else
        add entry (A_s + T, s) to the back of list A^DU
        add entry (A_s + 2T, s) to the back of list A^UU
return m
```

Algorithm 4: Pseudo-code for identical chambers, i.e. NLS-id, with unit capacity.

inserted in list $\mathcal{A}^{UU}$ have a time value of $A_s + 2T$ for some A_s, and all entries inserted in list $\mathcal{A}^{UD}$ have a time value of $A_s + T$ for some A_s. Each of the lists thus remains sorted by the time value of the contained entries at all times. Finding the earliest entry in two of the lists is then easily performed in constant time by comparing the first entry of each of the lists. Deleting the first entry as well as adding a new entry to the back of a list also require only constant time. Whenever an entry of one of the lists is iterated over, it is deleted. Since only $O(n)$ entries are added to lists throughout the procedure, iterating through the lists thus also takes $O(n)$ time in total. It follows that the entire procedure runs in linear time. We can summarize the discussion above as follows:

Theorem 5.4. *For the setting with identical chambers, a no-wait schedule can be found or shown not to exist, i.e. problem NLS-id can be solved, in $O(n)$ time.*

Since problem NLS-id is equivalent to the trapezoid graph colouring problem described above, the following result immediately follows.

Corollary 5.1. *Finding the chromatic number of a trapezoid graph where the trapezoids are identical up- or down-oriented isosceles triangles can be done in $O(n)$ time.*

5.7 An algorithm for arbitrary m

In the following, we propose a dynamic programming (DP) approach that solves the general problem NLS, stated in Section 5.1. That is, we consider an arbitrary number of chambers m, with non-identical lockage times T_c and capacities C_c for $c \in \mathcal{C}$. The proposed approach is similar to an $O(mn^{m+1})$ DP algorithm presented by Sung and Vlach (2005) for a parallel machine scheduling problem with deadlines and just-in-time jobs.

We assume ships to be numbered in non-decreasing order of arrival times. Ties are broken by letting ships arriving downstream have lower numbers than ships arriving upstream. Remaining ties are broken arbitrarily. Our DP approach assigns ships to chambers in increasing order of these numbers. We consider states $(s_1, \ldots, s_m)$ where s_c represents the last ship $1 \leq s_c \leq n$ that has been assigned to chamber c $(1 \leq c \leq m)$. Furthermore, we restrict ourselves to states where a ship is assigned to a chamber only if all earlier ships have also been assigned. Thus, in a given state $(s_1, \ldots, s_m)$, the first $\max_{c \in \mathcal{C}} s_c$ ships have been assigned to

chambers. We consider a transition from $(s_1, \ldots, s_m)$ to $(s'_1, \ldots, s'_m)$ if there is a $c^* \in \mathcal{C}$ such that:

1. $s'_{c^*} > s_{c^*}$ and $s'_c = s_c$ for each $c \in \mathcal{C}$ with $c \neq c^*$,
2. ships $(\max_c s_c) + 1, \ldots, s'_{c^*}$ travel in the same direction and arrive at the same time,
3. s'_{c^*} arrives at least T_{c^*} time units later than s_{c^*} if both travel in opposite direction and s'_{c^*} arrives at least $2T_{c^*}$ time units later than s_{c^*} if both travel in the same direction, and
4. $s'_c - (\max_c s_c) \leq C_{c^*}$.

This transition represents assigning ships $(\max_c s_c) + 1, \ldots, s'_{c^*}$ to chamber c^*. This is allowed only if chamber c^* is available after handling ship s_{c^*}. If more than one ship is assigned to chamber c^*, these ships must arrive simultaneously, travel in the same direction, and the chamber's capacity must not be exceeded. We consider an initial state $(0, \ldots, 0)$ representing that no ships are assigned to any chambers yet. The question is whether we can reach a state $(s_1, \ldots, s_m)$ with $\max_{c \in \mathcal{C}} s_c = n$ by any sequence of transitions.

In this DP we use $O(n^m)$ states and $O(mn^{m+1})$ transitions. However, we can further restrict the set of transitions by always assigning the maximum number of ships (up to the chamber's capacity) which travel in the same direction and arrive at the same time as ship $s_{c^*} + 1$. Each state then has m transitions: one per chamber. This leaves us with $O(mn^m)$ transitions, which also constitutes the runtime complexity. Note that the complexity is polynomially bounded if the number of chambers is fixed. We can conclude with the following theorem:

Theorem 5.5. *The problem setting with a fixed number of arbitrary chambers, i.e. problem NLS-m, can be solved in $O(mn^m)$ time.*

5.8 Number of chambers m part of the input

We prove here that problem NLS is NP-complete.

Theorem 5.6. *Deciding whether a no-wait solution exists for an arbitrary number of chambers, even when all ships travel in the same direction and arrival times are distinct, i.e. problem NLS-uni-distinct, is strongly NP-complete.*

Proof. We prove the theorem by a reduction from numerical matching with target sums where all given integers are distinct. We will denote this problem as dNMTS. Hulett et al. (2008) showed that this special case of the classical NMTS problem is strongly NP-complete. In an instance of dNMTS we are given $3n$ pairwise distinct positive integers a_i, b_i, c_i $(1 \leq i \leq n)$ with $\sum_i^n c_i = \sum_i^n (a_i + b_i)$. The question is whether there exists a collection of n triples (i, j, k) such that, for each triple, $a_i + b_j = c_k$, and such that each integer in the input occurs in exactly one triple. We show that we may impose, without loss of generality, two additional constraints on the instances of dNMTS. Our assumptions are (i) $\max_i a_i - \min_i a_i < \min_i b_i$, and (ii) $\max_i c_i - \min_i c_i < \min_i b_i$. Assumption (i) is immediately seen to hold since the instances in the proof provided by Hulett et al. (2008) satisfy $\max_i a_i < \min_i b_i$. For assumption (ii), observe that we may transform any instance into an equivalent instance where $\max_i c_i - \min_i c_i < \min_i b_i$ by adding an arbitrary positive K to all b_i and c_i of the original instance. This does not change the answer to the decision question; also note that by adding any $K > 0$, all resulting values remain distinct, and assumption (i) remains valid. Since the left-hand side of the inequality in assumption (ii) does not change by this operation, we obtain the desired result by choosing any $K > \max_i c_i - \min_i c_i - \min_i b_i$.

For any instance of dNMTS satisfying the two assumptions specified above, we now construct an instance of the lock scheduling problem with parallel chambers as follows. There are $2n$ ships arriving on the downstream side: n ships arrive at times $A_i = a_i$ and n ships arrive at times $A_{n+k} = c_k$, with $i, k \in \{1, \ldots, n\}$. For convenience, we will refer to the first n arrivals as the set of ships A, and the last n arrivals as the set of ships C. There are $m = n$ chambers, each with a lockage time equal to $T_j = {}^{b_j}/_2$ for $j \in \{1, \ldots, m\}$. The question remains whether a no-wait solution exists for this instance. Figure 5.14 shows a graphical representation. Next, we show that the given instance of dNMTS is a 'yes' instance if and only if there exists a no-wait solution to the constructed lock scheduling instance.

If there is a solution to the instance of dNMTS, each triple (a_i, b_j, c_k) corresponds to a combination of a chamber with one ship from the set A, and one ship from the set C. If the ship in A enters the chamber with lockage time ${}^{b_j}/_2$ at time a_i, the ship in C can enter the same chamber at time $c_k = a_i + 2\,({}^{b_j}/_2)$. Neither ship incurs any waiting time. Since each ship in A and C corresponds to exactly one such triple, there exists a no-wait solution.

On the other hand we argue that if a no-wait solution exists, there

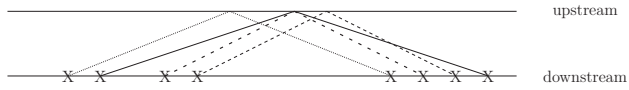


Figure 5.14: Graphical representation of the instance for the proof. Ship arrivals are marked with 'X', time passes from left to right. The tilted lines correspond to lock movements of the different chambers in a no-wait solution.

must also exist a corresponding set of triples that satisfies the requirement of the dNMTS problem. We first observe that in all no-wait solutions, each chamber handles exactly one ship from A, and one ship from C. Indeed, since all arrival times are distinct, a chamber cannot simultaneously transfer more than one ship without introducing waiting time. Further, it follows from $\max_i a_i - \min_i a_i < \min_i b_i$ that no chamber can serve two ships from A so that no ships incur waiting time. Similarly, it holds that no chamber can serve two ships from C without introducing waiting time. Thus, in a given no-wait solution, each chamber transfers exactly one ship from A and one ship from C. For each chamber j, let $i(j)$ be the index of the ship from A and $k(j)$ the index of the ship in C that is handled by chamber j. We thus obtain n triples $(a_{i(j)}, b_j, c_{k(j)})$. Since the solution is no-wait, we have $a_{i(j)} + b_j \leq c_{k(j)}$ for all j. Finally, because $\sum_i^n (a_i + b_i) = \sum_i^n c_i$, it is clear that if $a_{i(j)} + b_j < c_{k(j)}$ for any j, there must exist a j' such that $a_{i(j')} + b_{j'} > c_{k(j')}$, which would mean that at least one ship incurs waiting time. It follows that $a_{i(j)} + b_j = c_{k(j)}$ for all $j \in \{1, \ldots, n\}$, and there thus exists a set of triples that identifies the given instance of dNMTS as a 'yes' instance. This concludes the proof. □

As mentioned in the introduction, when viewing a chamber as a machine and an arrival as a job represented by one interval for each machine, each starting at the same moment in time, the above reduction shows that the problem considered by Böhmová et al. (2013) (called Interval Selection with cores) remains NP-complete even when all intervals that correspond to the same machine have the same length.

5.9 Conclusion

In this chapter, we considered the problem setting of a single lock consisting of parallel chambers. We analysed the complexity of finding no-wait

schedules. The resulting problem closely resembles an interval scheduling problem, and also relates to a graph colouring problem on unit interval graphs. By investigating different problem variants of this setting, we were able to extend and improve a number of known results from literature. We covered results for the setting with two locks, the setting with identical locks, and the general setting with arbitrary chambers. Additionally, we showed that the general problem setting of deciding whether a no-wait schedule exists is strongly NP-complete.

An obvious extension which could be further investigated is to consider, for all instances where no no-wait solution exists, the problem of finding a feasible schedule that minimizes the total waiting time. A different objective could be to minimize the maximum waiting time. As for a single-chamber lock, it may be interesting to consider problem settings where there is uncertainty on the input parameters, or to look at the on-line setting. See Chapter 6 for a brief description of these problem extensions.

Chapter 6

General conclusion and future research

In the preceding chapters, we have discussed different facets of the lock scheduling problem. Inspired by the practical problem of scheduling locks, and locks on inland waterways in particular, we have investigated a variety of the underlying scheduling problems from a mathematical point of view. In Chapter 1, we described some context and motivation for this problem, and provided an overview of related literature. Further, we defined some terminology and introduced a notation in order to allow more formal problem descriptions. We then focussed on the following problem settings. Chapter 2 described a basic problem concerning a single lock consisting of a single chamber. This problem, as well as a number of extensions, can be solved in polynomial time. In Chapter 3, we investigated the computational complexity of minimizing the total waiting time for multiple locks arranged in a sequence, and showed that this problem is NP-hard. Furthermore we showed that, for certain special cases, there exists a so-called synchronised solution which is optimum. The hardness result obtained for locks in sequence suggests an integer programming approach. Chapter 4 introduced two such integer programming models and evaluated their performance. The trade-off between the objective of minimizing flow time and minimizing emissions was also investigated. Finally, Chapter 5 considered the problem setting with multiple chambers arranged in parallel, and investigated the computational complexity of finding no-wait schedules.

We conclude by identifying some possible directions for further research which are not specific to any of the individual chapters. One direction for future research could relax the assumption that all input data are known initially. We describe two ways in which this is possible. First, stochastic effects could be considered, since a number of parameters which were assumed to be known may in fact be subject to different influences which cannot be accurately predicted. Notably, the observed arrival times may differ from the expected arrival times that were predicted or communicated at an earlier point in time.

A different approach is to maintain the assumption of deterministic arrival times, but to consider the on-line setting, i.e. assume that arrival information only becomes available as time progresses, so that it may be interesting to update the current schedule as more information becomes available. One particular setting, which better reflects practical lock scheduling problems, compromises between the off-line and on-line settings. It can be reasonably expected that arriving ships are announced at least some time before their arrival. The result is a limited 'look-ahead' interval where all arrival times are assumed to be known. Clearly, an approach with such a look-ahead interval integrates well with a 'rolling horizon' procedure, where new instances are evaluated over time as more information becomes available; the schedule can then also be updated as new solution are obtained.

A more general extension of the research presented here would be to consider a network of waterways, where there may exist multiple 'routes' from a ship's origin to its destination. This setting may be of particular interest if certain segments of the network are heavily congested, either due to the traffic density or due to a technical outage.

Integrating the scheduling of locks within more general waterway management may also be valuable. As an example, the operational planning for locks could be taken into account while a desired water level is actively maintained, for example during a dry season; the scheduling of locks could then integrate with the operational planning of pumps or hydroelectric power plants along a waterway.

It may also be possible to apply some of the results in related settings such as, for example, the scheduling of bridges or traffic intersections. It should be noted, however, that there is no exact correspondence between these alternative applications and the scheduling of locks.

Bibliography

Aggarwal, A. and Park, J. K. (1993). Improved algorithms for economic lot size problems. *Operations Research*, 41:549–571.

Antwerp Port Authority (2011). Port of Antwerp starts building the largest lock in the world. `http://www.deurganckdoksluis.be/en/press/port-antwerp-starts-building-largest-lock-world`. (accessed October 7, 2011).

Aspvall, B., Plass, M. F., and Tarjan, R. E. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121 – 123.

Baptiste, P. (2000). Batching identical jobs. *Mathematical Methods of Operations Research*, 52:355–367.

Böhmová, K., Disser, Y., Mihalák, M., and Widmayer, P. (2013). Interval selection with machine-dependent intervals. In *WADS'13*, pages 170–181.

Boudhar, M. (2003). Scheduling a batch processing machine with bipartite compatibility graphs. *Mathematical Methods of Operations Research*, 57:513–527.

Boulter, P. G. (2007). ARTEMIS: Assessment and reliability of transport emission models and inventory systems: final report. Technical report, TRL limited.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., and van de Velde, S. L. (1998). Scheduling a batching machine. *Journal of Scheduling*, 1:31–54.

Brucker, P., Knust, S., Cheng, T. E., and Shakhlevich, N. V. (2004). Complexity results for flow-shop and open-shop scheduling problems

with transportation delays. *Annals of Operations Research*, 129(1):81–106.

Caris, A., Janssens, G., and Macharis, C. (2007). A simulation approach to the analysis of intermodal freight transport networks. In *ESM'2007 Proceedings*. EUROSIS.

Central Commission for Navigation on the Rhine (2015). Electronic ship reporting in inland navigation. Leaflet.

Central Commission for the Navigation of the Rhine (2011). Inland AIS. Leaflet.

Cheng, T. C. E., Yuan, J. J., and Yang, A. F. (2005). Scheduling a batch-processing machine subject to precedence constraints, release dates and identical processing times. *Computers and Operations Research*, 32:849–859.

ChinaDaily (2011). Three gorges ship lock marks 8 years of operation. `http://usa.chinadaily.com.cn/china/2011-06/18/content_12730491.htm`. (accessed October 7, 2011).

Condotta, A., Knust, S., and Shakhlevich, N. V. (2010). Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13:463–477.

Disser, Y., Klimm, M., and Lübbecke, E. (2015). Scheduling bidirectional traffic on a path. *Computing Research Repository*, abs/1504.07129.

Du, J. N. and Yu, S. M. (2003). Dynamic programming model and algorithm of shiplock scheduling problem. *Computer and Digital Engineering*, 31:47–50. (in Chinese).

Emmons, H. and Vairaktarakis, G. (2013). *Flow Shop Scheduling*, volume 182 of *International series in Operations Research & Management Science*. Springer US.

European Commission (2012). Towards "NAIADES II", promoting, greening and integrating inland waterway transport in the single EU transport area. Technical report, European Commission.

European Commission (2013). Towards quality waterway transport. Memo, European Commission.

European Commission (2015). Promotion of inland waterway transport. `http://ec.europa.eu/transport/inland/promotion/promotion-en.htm`. (accessed November 21, 2015).

Even, S., Itai, A., and Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703.

Fagerholt, K., Laporte, G., and Norstad, I. (2010). Reducing fuel emissions by optimizing speed on shipping routes. *Journal of the Operational Research Society*, 61(3):523–529.

Federgruen, A. and Tzur, M. (1991). A simple forward algorithm to solve dynamic lot sizing models with n periods in $O(n \log n)$ or $O(n)$ time. *Management Science*, 37:909–925.

Felsner, S., Müller, R., and Wernisch, L. (1997). Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32.

Finke, G., Jost, V., Queyranne, M., and Sebő, A. (2008). Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, 156:556–568.

Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press.

Frank, O. (1966). Two-way traffic on a single line of railway. *Operations Research*, 14(5):801–811.

Gafarov, E. R., Dolgui, A., and Lazarev, A. A. (2015). Two-station single-track railway scheduling problem with trains of equal speed. *Computers & Industrial Engineering*, 85:260–267.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267.

Goodban Belt LLC (2010). New york state canal system - modern freightway. Technical report, New York State.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In Hammer, P. L., Johnson, E. L., and Korte,

B. H., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier.

Gupta, U., Lee, D., and Leung, J. Y.-T. (1979). An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, C-28(11):807–810.

Hattendorf, J. B. (2007). Rivers, canals, and inland waterways. In Hattendorf, J. B., editor, *The Oxford Encyclopedia of Maritime History*. Oxford University Press.

Hermans, J. (2014). Optimization of inland shipping - a polynomial time algorithm for the single ship single lock optimization problem. *Journal of Scheduling*, 17:305–319.

Hulett, H., Will, T. G., and Woeginger, G. J. (2008). Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36:594–596.

Hvattum, L. M., Norstad, I., Fagerholt, K., and Laporte, G. (2013). Analysis of an exact algorithm for the vessel speed optimization problem. *Networks*, 62(2):132–135.

Kolen, A. W., Lenstra, J. K., Papadimitriou, C. H., and Spieksma, F. C. (2007). Interval scheduling: A survey. *Naval Research Logistics*, 54(5):530–543.

Krumke, S. O., Thielen, C., and Westphal, S. (2011). Interval scheduling on related machines. *Computers & Operations Research*, 38(12):1836–1844.

Kunst, M. (2013). Organisation of vessel traffic management centres of the future. In *Smart Rivers Conference 2013 Abstract Booklet*, Liege, Belgium.

Lee, C., Uzsoy, R., and Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40:764–775.

Lübbecke, E., Lübbecke, M., and Möhring, R. H. (2014). Ship traffic optimization for the Kiel Canal. Technical Report 4681, Optimization Online.

Luy, M. (2012). Ship lock scheduling. `http://sourceforge.net/projects/lockscheduling/`. (accessed November 21, 2015).

Meijer, F. (2007). Trading vessels: ancient vessels. In Hattendorf, J. B., editor, *The Oxford Encyclopedia of Maritime History*. Oxford University Press.

Nauss, R. M. (2008). Optimal sequencing in the presence of setup times for tow/barge traffic through a river lock. *European Journal of Operational Research*, 187:1268–1281.

Ng, C. T., Cheng, T. C. E., Yuan, J. J., and Liu, Z. H. (2003). On the single machine serial batching scheduling problem to minimize total completion time with precedence constraints, release dates and identical processing times. *Operations Research Letters*, 31:323–326.

Notteboom, T. (2007). Inland waterway transport of containerised cargo: From infancy to a fully fledged transport mode. *Journal of Maritime Research*, 4(2):63–80.

Panama Canal Authority (2016). A message from the Panama Canal Authority. The Canal Connection, March.

Passchyn, W., Briskorn, D., and Spieksma, F. C. R. (2016a). Mathematical programming models for lock scheduling with an emission objective. *European Journal of Operational Research*, 248(3):802 – 814.

Passchyn, W., Briskorn, D., and Spieksma, F. C. R. (2016b). No-wait scheduling for locks. Technical Report KBI_1605, KU Leuven, Research group Operations Research and Business Statistics, Leuven, Belgium.

Passchyn, W., Coene, S., Briskorn, D., Hurink, J. L., Spieksma, F. C. R., and Vanden Berghe, G. (2016c). The lockmaster's problem. *European Journal of Operational Research*, 251(2):432 – 441.

Petersen, E. R. and Taylor, A. J. (1988). An optimal scheduling system for the Welland Canal. *Transportation Science*, 22:173–185.

Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249.

Prandtstetter, M., Ritzinger, U., Schmidt, P., and Ruthmair, M. (2015). A variable neighborhood search approach for the interdependent lock scheduling problem. In Ochoa, G. and Chicano, F., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 9026 of *Lecture Notes in Computer Science*, pages 36–47. Springer International Publishing.

Promotie binnenvaart Vlaanderen (2013). Cijfers met vaart. Leaflet. (In Dutch).

Psaraftis, H. N. and Kontovas, C. A. (2013). Speed models for energy-efficient maritime transportation: A taxonomy and survey. *Transportation Research Part C: Emerging Technologies*, 26(0):331 – 351.

Qian, J. and Eglese, R. (2014). Finding least fuel emission paths in a network with time-varying speeds. *Networks*, 63(1):96–106.

Rijkswaterstaat (2010). Twentekanalen: uitbreiding sluis eefde. `http://www.rijkswaterstaat.nl/water/projectenoverzicht/twentekanalen-uitbreiding-sluis-eefde/`. (in Dutch, accessed October 7, 2011).

Ronen, D. (1982). The effect of oil price on the optimal speed of ships. *Journal of the Operational Research Society*, 33(11):pp. 1035–1040.

Sack, J. R. and Urrutia, J., editors (2000). *Handbook of Computational Geometry*. North-Holland - Elsevier, Amsterdam, The Netherlands.

Smith, L. D., Nauss, R. M., Mattfeld, D. C., Li, J., Ehmke, J. F., and Reindl, M. (2011). Scheduling operations at system choke points with sequence-dependent delays and processing times. *Transportation Research Part E*, 47:669–680.

Smith, L. D., Sweeney, D. C., and Campbell, J. F. (2009). Simulation of alternative approaches to relieving congestion at locks in a river transportation system. *Journal of the Operational Research Society*, 60:519–533.

Sung, S. C. and Vlach, M. (2005). Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, 8(5):453–460.

Ting, C. and Schonfeld, P. (1999). Effects of speed control on tow travel costs. *Journal of waterway, port, coastal, and ocean engineering*, 125(4):203–206.

Ting, C. and Schonfeld, P. (2001). Control alternatives at a waterway lock. *Journal of waterway, port, coastal, and ocean engineering*, 127(2):89–96.

US Army Corps of Engineers (2014). Waterborne commerce from the united states. Technical report, US Army Corps of Engineers Navigation Data Center.

van Haastert, M. (2003). Planningsmodel scheepvaartafhandeling bij de noordersluis in IJmuiden. Master's thesis, TU Delft. (in Dutch).

Verstichel, J. (2013). *The Lock Scheduling Problem*. PhD thesis, KU Leuven.

Verstichel, J., De Causmaecker, P., Spieksma, F. C. R., and Vanden Berghe, G. (2014a). Exact and heuristic methods for placing ships in locks. *European Journal of Operational Research*, 235(2):387–398.

Verstichel, J., De Causmaecker, P., Spieksma, F. C. R., and Vanden Berghe, G. (2014b). The generalized lock scheduling problem: An exact approach. *Transportation Research E, Logistics and Transportation Review*, 65:16–34.

Verstichel, J. and Vanden Berghe, G. (2009). A late acceptance algorithm for the lock scheduling problem. *Logistik Management*, 5:457–478.

Wagelmans, A., Van Hoesel, S., and Kolen, A. (1992). Economic lot sizing: An $O(n \log n)$ algorithm that runs in linear time in the wagner-whitin case. *Operations Research*, 40:S145–S156.

Waterwegen en Zeekanaal NV and nv De Scheepvaart (2014). *Masterplan voor binnenvaart op de Vlaamse waterwegen - Horizon 2020*. Willebroek: Waterwegen en Zeekanaal NV, Hasselt: nv De Scheepvaart. (in Dutch).

Webster, S. and Baker, K. R. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, 43:692–703.

Zhao, Z. J., Lau, H. C., and Ge, S. S. (2009). Integrated resource allocation and scheduling in a bidirectional flowshop with multimachine and COS constraints. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(2):190–200.

Doctoral dissertations from the Faculty of Economics and Business

Doctoral dissertations from the Faculty of Economics and Business, see:
http://www.kuleuven.ac.be/doctoraatsverdediging/archief.htm.