KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

The lock scheduling problem

Jannes Verstichel

Dissertation presented in partial
fulfillment of the requirements for the
degree of Doctor in Engineering

November 2013

The lock scheduling problem

Jannes VERSTICHEL

Supervisory Committee:
Prof. dr. ir. Pierre Verbaeten, chair
Prof. dr. Patrick De Causmaecker, supervisor
Prof. dr. ir. Greet Vanden Berghe, co-supervisor
Prof. dr. Frits C.R. Spieksma
Prof. dr. ir. Dirk Cattrysse
Prof. dr. Marc Denecker
Prof. dr. Gerhard Wäscher
(Otto-von-Guericke-Universität Magdeburg, Germany)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

November 2013

D/2013/7515/129
ISBN 978-94-6018-744-5

Preface

What do you think about starting a Ph.D. under my supervision after you graduate?
- I'd love to! But only if it is practically oriented, I like to be able to 'feel' what I'm working on...

Admittedly, despite my enthusiasm, dedicating four long years of my life to a single subject seemed a little frightening at first. Fortunately enough, the venture turned out to be a most interesting and gratifying experience. The years flew while I was 'playing' with locks and ships under the excellent guidance of my supervisors Patrick De Causmaecker and Greet Vanden Berghe. It is with their support that I managed to bring this blend of theoretical research and real-life applications to a good end.

When it comes to the practical parts of this thesis, I owe a great deal to Marnix Delee and Ive Braspenningx. Under the motto *nothing ventured, nothing gained* I sent an e-mail through the contact-page on the Port of Antwerp's website on the last day before the holidays, hoping to get a reply by the time I got back from vacation. I got one within hours, and a few days later I paid my first of many visits to the port. I cannot thank you enough for giving such an opportunity to an unknown doctoral student. It is thanks to your support and enthusiasm that I reached results I did not dare dream of achieving when I started this Ph.D.

Also many thanks to Theo Blommaert and Peter Scherpenisse from the Rijkswaterstaat. Your enthusiasm when I presented my research was a real boost for my confidence.

I would like to thank Frits Spieksma, Dirk Cattrysse, Marc Denecker, Gerhard Wäscher and Pierre Verbaeten, the members of my jury and advisory committee, for their constructive and helpful remarks which helped improving the quality of my thesis.

Despite all intellectual challenges and interesting research directions, working on a single topic for four years can easily turn into a dull and lonely experience. Not so with the colleagues from CODeS at KAHO Sint-Lieven. Always ready for a discussion (coffee anyone?) they formed a top-notch sounding board, and thanks to their entertainment and support even the toughest days got a silver lining. So thank you Tony Wauters, Wim Vancroonenburg, Pieter Smet, Joris Maervoet, Joris Kinable, Jan Christiaens, Peter Demeester, Tim Vermeulen, Katja Verbeeck, Mihail Mihaylov and Sam Vanmalderen. Special thanks also go to Erik Van Achter and Luke Connolly for improving the quality of my academic writing.

However fascinating the research has been, living of enthusiasm alone is impossible. I therefore would like to thank IWT Vlaanderen for providing me with a steady income during the past four years, making my life a lot easier.

A significant part of this thesis was written while a little adventurer was exploring, producing the necessary exclamations of happiness and wonder, the world at my feet. Juta managed to turn writing this thesis into a pleasant and unforgettable experience. Last but not least, I would like to thank my better half Sabien and my amazing parents, without whom none of this would have been possible. So, from the bottom of my heart, thank you!

Abstract

The present thesis introduces the lock scheduling problem and promising decompositions into sub problems. Several combinatorial optimization methods have been developed for the lock scheduling problem. Single and parallel chamber locks and lock operations in both an inland and mixed-traffic setting are considered, and a mathematical model precisely describing the problem is presented. Three interrelated sub problems can be discerned: ship placement, chamber assignment and lockage operation scheduling. These are closely related to the two dimensional bin packing problem, the assignment problem and the (parallel) machine scheduling problem respectively. After an in-depth analysis the ship placement sub problem is decomposed by exploiting its sequence constraints, and both an exact and a heuristic approach are developed and tested on a large and diverse test set. A decision support tool for lock masters is presented and evaluated during live-tests at mixed-traffic locks in a major port and on an important waterway. These tests reveal that the introduction of fast and high-quality optimization software in the lock master's tool suite can increase lock efficiency by enabling a faster reaction to last-minute changes, quickly producing good solutions during peak traffic and increasing the lock's planning horizon. The lock scheduling problem is solved through Combinatorial Benders' decomposition by combining the assignment and scheduling problems into a master problem and considering ship placement as a sub problem. Efficient cut separation methods are introduced and tested and the performance of an exact and of a heuristic solution method for the packing sub problem are evaluated. Experiments show that the decomposition method outperforms a monolithic branch-and-bound approach, and that the slow convergence of the master problem is currently the decomposition method's limiting factor.

Beknopte samenvatting

Deze thesis introduceert het sluisplanningsprobleem voor sluizen met binnenlands verkeer en in havenomgevingen waar zowel binnen- als zeevaart aanwezig zijn. Een accuraat mathematisch model voor dit gegeneraliseerde sluisplanningsprobleem is ontwikkeld en getest op een dataset bestaande uit gegenereerde en historische data. Het sluisplanningsprobleem bestaat uit drie geconnecteerde subproblemen: schip positionering, kolktoewijzing en het plannen van schuttingen. Deze problemen zijn sterk gerelateerd aan het twee dimensionale *bin packing* probleem, het *assignment* probleem en het *(parallel) machine scheduling* probleem respectievelijk. Na een diepgaande analyse wordt het schip positionering subprobleem opgesplitst door gebruik te maken van volgordebeperkingen, en worden exacte en heuristische oplossingsmethoden ontwikkeld en getest op een grote en diverse dataset. Het onderzoek vormt de basis van een beslissingsondersteunende tool die door sluiswachters geëvalueerd is tijdens live-tests op sluizen voor gemengd verkeer. Deze tests tonen aan dat de toevoeging van performante optimalisatiesoftware aan de huidige tools van de sluiswachters de efficiëntie van sluizen kan verhogen: door snel hoog kwalitatieve oplossingen te berekenen, kan er gereageerd worden op last-minute veranderingen, kan de aanwezige capaciteit beter benut worden tijdens piekmomenten en kan de planningshorizon van de sluizen verlengd worden. Het sluisplanningsprobleem wordt vervolgens opgelost via *Combinatorial Benders'* decompositie door het kolktoewijzingsprobleem en het plannen van schuttingen te beschouwen in een *master problem* en de positionering van de schepen in een *sub problem* aan te pakken. Efficiënte *cut separation* methodes worden geïntroduceerd en getest, en de performantie van zowel exacte als heuristische oplossingsmethoden wordt geëvalueerd. De experimenten tonen aan dat deze decompositiemethode performanter is dan een monolitische *branch-and-bound* methode, en dat de trage convergentie van het master problem momenteel de beperkende factor is van deze oplossingsmethode.

Abbreviations

LSP	Lock Scheduling Problem
SPP	Ship Placement Problem
2D	Two-dimensional
SBSBPP	Single bin size bin packing problem
MOGLi	Multilevel Optimization and Generic models for Lock operations
MP	(Restricted) Master Problem
SP	Sub Problem
MTE	Million Tonnes Equivalent
UMR	Upper Mississippi River
MIP	Mixed Integer Programming
FCFS	First-come-first-served (based on arrival time)
FIFO	First-in-first-out (based on index in an ordered set)
MILP	Mixed Integer Linear Programming
SSC	Single Small Chamber lock
SLC	Single Large Chamber lock
PSC	Parallel Small Chamber lock
PLC	Parallel Large Chamber lock
MCT	Multi Chamber Type lock
BE	Berendrecht lock
ZV	Zandvliet lock
BO	Boudewijn lock
VC	Van Cauwelaert lock
RO	Royers lock
KL	Kallo lock
MIS	Minimal Infeasible Subsets

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Both ports and waterways are under ever increasing pressure due to growing maritime traffic and transport. Handle times are constantly reduced while flexibility is increased to maintain or improve market shares. While many aspects of handling ships and containers in seaports have been extensively researched [Stahlbock and Voß, 2008], locks, a key component of a tide independent port's infrastructure, have been utterly neglected in academia. Locks constitute a complex optimization problem: the vast number of ships entering and leaving the harbor on a daily basis must be assigned to lock chambers, their exact position inside the locks has to be defined and the resulting lockages require scheduling. A ship's handling time can strongly increase from a lock's suboptimal usage. The lock's incapacity to transfer a given ship in time might lead to missing its time window at the terminal, resulting in great inefficiency: the ship's total time in port increases and the terminal operations are disturbed. Similarly, inland waterways need to reduce the waiting times at locks to a minimum to increase the share of waterway transport in multimodal transporation[1] [European Commission, 2009, 2011]. Inland navigation is a most promising transport mode in the multimodal chain due to its excess capacity in the network and environmentally friendly nature. In Western Europe, and especially Belgium, inland waterways play a crucial role in hinterland access of major sea ports [Notteboom and Rodrigue, 2005]. The current increase of barge traffic and its future prospects make it of paramount importance to reduce ships' waiting

[1] Multimodal transportation is the combination of multiple transport modes in a single transport chain without a change of container for the goods.

times, thus enabling inland navigation to compete with road transportation. However, periods of drought force the lock operators to minimize the number of lockage operations (i.e. water usage) to transfer these ships. Currently human experts schedule locks with little or no support from optimization software.

Against the backdrop of these facts, the aim of the present doctoral dissertation is to formulate exact and heuristic optimization methods for tackling the *Lock Scheduling Problem* (LSP). The LSP encompasses three strongly interconnected sub problems: an assignment, a packing, and a scheduling problem, and therefore decomposition methods will be of special interest. The development of a decomposition method, based on existing exact decomposition approaches applicable in both exact and heuristic frameworks can be considered the secondary goal of this thesis. Although many exact decomposition methods exist and have been researched in a multitude of papers, the knowledge behind these methods has rarely been transferred into practical applications and heuristic solution approaches. With an accessible explanation of how one such method can be applied to real-life LSP instances and comparable structured combinatorial optimization problems, this thesis can thus also be read as an attempt to transfer this knowledge to heuristic and applied research communities.

1.2 Structure of the thesis

Chapter 2 begins by sketching the LSP in ports and on inland waterways. The similarities and differences between both environments are clarified, and the impact of the LSP on the maritime transportation chain is further elucidated. The chapter continues with an extensive literature review on the LSP, port operations and inland navigation. A mathematical model for the LSP enables computing optimal solutions to (very) small LSP instances in both port and inland settings, and functions as a reference for other solution approaches.

Chapter 3 focusses on the *Ship Placement Problem* (SPP), which is the packing sub problem of the LSP. First, the SPP and its interaction with the LSP are defined, after which the problem is identified as a variant of the *two-dimensional rectangular single bin size bin packing problem* (2D rectangular SBSBPP) [Wäscher et al., 2007]. After reviewing several potential solution methods for the SPP, a mathematical model for the SPP is introduced. Next, a decomposition scheme is introduced, enabling the transformation of a multi-lockage SPP into a series of single lockage SPP's which can be solved efficiently using exact or heuristic methods. Although the resulting exact solution method allows for reasonably fast computations in a controlled environment, real-life applications of the SPP require a much faster and predictable response time.

The multi-order best-fit, which is an extension of the best-fit heuristic for the 2D orthogonal strip packing problem [Burke et al., 2004], combines such fast and stable computation time with a low optimality gap. The chapter concludes with a thorough computational study of the solution approaches for the SPP.

Chapter 4 reports on the real-life application of MOGLi[2], a decision support tool for lock masters. The decision support software proposes one or more possibilities for executing the next lockage(s) based on a list of arriving ships and the selected lock configuration. Ships capable of being transferred together are automatically selected and positioned in a feasible and easy to understand way. In the case of multi-chamber locks, the lock master can quickly evaluate the effect of an alternative chamber for transferring some of the arriving ships. The tool is highly configurable and easily applied to locks both in major ports and on inland waterways. MOGLi was tested during during a total of four one-day live-tests on the Boudewijn-Van Cauwelaert lock complex in the port of Antwerp and at the locks of Terneuzen (The Netherlands) during which it successfully assisted the lock masters in their daily operations.

The lock scheduling problem is decomposed into a *restricted master problem* (MP) and a *sub problem* (SP) in Chapter 5. The presented decomposition scheme facilitates interaction between the MP and the SP through the generation of combinatorial Bender's cuts. The master problem first assigns the ships to lock chambers, after which it attempts to schedule the lockages. The sub problem handles the positioning the ships inside the lock chambers. Whenever the sub problem identifies an infeasible lockage, i.e. a set of ships that cannot be transferred simultaneously due to the chamber's capacity or safety constraints, combinatorial inequalities (cuts) are generated and added to the master problem. The master problem and sub problem are solved iteratively until a provable optimal (and feasible) schedule is obtained. When the MP and/or the SP are solved heuristically, another stopping criterion can be applied. Several cut separation methods are proposed and compared in an extensive computational study.

In Chapter 6 the main contributions of the doctoral research in the fields of lock scheduling, exact and heuristic decomposition and applications of operations research are summarized. The dissertation ends with some suggestions for interesting future research directions.

Appendix A discusses an extension of the best-fit heuristic for the orthogonal strip packing problem from Burke et al. [2004]. The resulting three-way (best-fit) heuristic served as the base for the multi-order best-fit heuristic for the ship placement problem from Chapter 3, and was published in Verstichel et al. [2013c]. The appendix presents the developed best-fit extensions, several of

[2]Multilevel Optimization and Generic models for Lock operations

which were applied in the multi-order best-fit heuristic, and contains a detailed explanation of how the (multi-order) best-fit heuristic works. This chapter has been added as an appendix to allow for a main text that is focussed entirely on the lock scheduling problem.

A short overview of the graphical user interface and configuration options of MOGLi is added in Appendix B.

Chapter 2

The generalized lock scheduling problem

The lock scheduling problem (LSP) is introduced, and a mathematical model for the LSP is presented. To begin, we sketch the LSP as it is encountered on the Albertkanaal and on other inland waterways. We then frame the LSP in a mixed-traffic environment like the Port of Antwerp. An extensive literature review on the LSP, inland navigation and relevant port operations is provided in Section 2.3. Section 2.4 continues with a formal description of the LSP, followed by a mathematical model in Section 2.5. Some improvements to the model are presented in Section 2.6. A computational study of the presented model is added in Section 2.7, which demonstrates the applicability of the mathematical model to very small LSP instances only. Parts of this chapter were published as Verstichel and Vanden Berghe [2009] and Verstichel et al. [2013b].

2.1 The inland lock scheduling problem

Barges travelling on a network of inland waterways often have to pass several locks. The locks control the water level and the flow on the inland waterways and overcome height differences in the landscape. A lock consists of at least one chamber in which barges can be transferred from one water level to another. When more than one chamber is available, the chambers can be paired (i.e. operated together) or operated independently. While some multi-chamber locks consist of several identical chambers, others have chambers of different

dimensions and properties. Depending on the size of the chamber, one or more vessels can be transferred together in a single lockage operation. Processing a ship in a lock may thus require up to three decisions, each with a significant impact on the quality of service: selecting the chamber that will transfer the ship, determining a position for the ship and setting a starting time for the lockage operation.

The large and well-connected network of inland waterways in Belgium provides ample applications of locks in all configurations and sizes. One example being the Albertkanaal, an important inland waterway connecting the port of Antwerp with the port of Liège. Over the years numerous industrial activities have emerged on its banks, with a total of over 37 *Million Tonnes Equivalent* (MTE) of processed cargo in 2012 [nv De Scheepvaart, 2012]. Six triple-chamber locks are used to overcome the height difference of 56m between Antwerp and Liège. Each lock consists of two identical small chambers and a single large chamber, all of which can be operated independently, and has a height difference between 5.7m and 14m. On the Albertkanaal, the increase of barge traffic and recent periods of drought make it of utmost importance to optimize the lock operations by reducing the number of lockage operations (i.e. water usage) and the waiting time of ships. A top-view of the Wijnegem lock on the Albertkanaal is added in Figure 2.1.

Another example is the Strépy-Thieu boat lift on the Canal du Centre (Figure 2.2). This boat lift is the largest in the world, with a height difference between the upstream and downstream reaches of 73 meters. The lock consists of two independent and watertight mobile cages (chambers) that are pulled up or lowered by the combined efforts of a counterbalance and four electrical motors [Walloon Government, 2013a]. This is a completely different approach from that of regular locks, where the water level inside the chamber is changed instead of moving the chamber (and its contents) from one water level to another. This boat lift has facilitated a large boost of the maritime traffic on the Canal du Centre, quadrupling the transported cargo from 0.2 MTE on the old lock system in 2001 to 0.8 MTE in 2003 when the new lift was fully operational. Roughly 1.1 MTE was transported over the boat lift in 2011 [Walloon Government, 2013b].

Locks are quite common in other European countries like The Netherlands, Germany and Great Britain as well. We will also briefly discuss two examples of inland locks outside of Europe: the Three Gorges Dam in China and the *Upper Mississippi River* (UMR) in the United States.

Figure 2.1: Photograph of the Wijnegem lock on the Albertkanaal in Belgium. (©2009 Google, TerraMetrics)

Figure 2.2: Photograph of the boat lift of Strépy-Thieu in Belgium. (©2006 Jean-Pol Grandmont)

2.2 The mixed-traffic lock scheduling problem

The port of Antwerp (Belgium), one of the largest in Europe, processed more than 180 MTE of cargo and seventy thousand ships in 2012, averaging at almost 200 ships per day [Port of Antwerp, 2012]. The harbor is situated at the river Scheldt with tidal differences averaging five meters and acts as a major hub for both inland and intercontinental cargo traffic. To ensure a constant water level at the harbor's docks, they are separated from the main water way by a number of locks with chamber sizes ranging from 180m x 22m to 500m x 68m. Figure 2.3 shows the Boudewijn-Van Cauwelaert lock complex in the port of Antwerp.

The mixed-traffic LSP (e.g. in the port of Antwerp) and the previously introduced inland LSP differ considerably. First and foremost, the maritime traffic in a port environment consists of several ship types, including barges, seagoing vessels, tugboats, etc. Each ship type requires specialist management and transferring several ship types in a single lockage operation impacts both the ship placement and the lockage scheduling sub problems. Contrary to the inland setting, safety distances are ship-dependent and cannot be assumed part of the ship's and the chamber's dimensions. The minimal safety distances between seagoing vessels depend upon their dimensions and tugboat requirements, while safety distances between seagoing vessels and barges depend on the size of the barge only. Additionally, several mooring restrictions may exist, preventing barges from mooring to a seagoing vessel, seagoing vessels from mooring to each other, etc. The presence of large seagoing vessels influences the scheduling part of the LSP: seagoing vessels may require a certain pre and post-processing time for entering/leaving the lock, (un)mooring at the quay and (dis)engaging the tugboats. As the chambers of port locks are typically larger than those from an inland setting, barges can also influence the lockage duration: when large numbers of barges are transferred together, the accumulated manoeuvring time may become significant.

Several other ports, like the ones of Ghent and Terneuzen, are (partially) situated behind a system of locks. A very famous example of mixed-traffic lock systems is found on the Panama Canal, which is an important waterway for international maritime trade.

Figure 2.3: Photograph of the Boudewijn - Van Cauwelaert lock complex in the Port of Antwerp, Belgium. (©2012 Google, Aerodata International Surveys, DigitalGlobe, GeoEye, Cnes/Spot Image)

2.3 Literature review

Only a small number of academic papers focus on lock planning from an operations research point-of-view. Wilson [1978] investigates the applicability of different queuing models for lock capacity analysis. The research shows that good queuing models exist for single chamber locks, but not for locks with parallel chambers. Coene and Spieksma [2011] study the lockmaster's problem, which focusses exclusively on the scheduling sub problem, assuming chambers with infinite capacity. They identify it as a batch scheduling problem that can, under certain conditions, be solved in polynomial time using a dynamic programming algorithm. Other lock scheduling research focuses on the Upper Mississippi River (UMR), where barges are joined together into tows for transport, which must be transferred by single chamber locks that are often smaller than the tow itself. The tow is split into different groups of barges and these groups are transferred sequentially, after which they are rejoined for the next phase of their travel. Nauss [2008] presents optimal sequencing of tows/barges for single chamber locks with set-up times. The approach enables one tow/barge to be transferred at a time. Furthermore, it considers all tows/barges to be present at the lock before the first lockage. A simulation model for comparing

different strategies to relieve congestion problems on the Upper Mississippi River is presented in Smith et al. [2009]. The strategies aim at increasing the throughput of the locks and a simulation tool was built for validating them. Smith et al. [2011] further increase the performance of these locks using more complex decision rules based on heuristics and *mixed integer programming* (MIP) models. The Three Gorges dam and Gezhouba dam have also been researched from a lock scheduling point of view. The Gezhouba lock consists of three independently operated identical parallel chambers, while the lock at Three Gorges consists of two single-direction multi-stage chambers. Zhang et al. [2008] consider navigation co-scheduling of the locks at both dams, compare it to flexible manufacturing systems and present a non-linear mathematical model. They consider a simplified version of the ship placement problem, taking into account standard 2D rectangular bin packing constraints only. A hybrid simulated annealing meta heuristic is developed and the approach is validated on historical data. Wang et al. [2013] consider the parallel single-direction locks at the Three Gorges dam and present a non-linear mathematical model based on integral calculus where the ship placement problem is considered equal to 2D rectangular bin packing. The convergence of ant colony optimization on this problem and accuracy of the available historical data are analysed.

Optimization and decision support for lock scheduling is an emerging research field in academics. While several aspects of the lock scheduling problem have been considered in literature, several hiatuses remain. The most obvious example is the absence of research considering seagoing vessels and mixed traffic. Furthermore, the ship placement problem (if considered at all) was reduced to 2D rectangular bin packing, thereby ignoring a wide array of operational limitations. Finally, a mathematical model for the lock scheduling problem with single or parallel chambers and mixed traffic is also missing.

2.4 Problem definition

The lock scheduling problem can be described as follows. A number of ships N need to traverse a lock, either in the upstream/incoming, or in the downstream/outgoing direction. The major components of the lock scheduling problem are visualised in Figure 2.4, which depicts a multi chamber lock transferring several ships. These ships constitute the first major component of the lock scheduling problem. The sets of upstream and downstream ships are denoted N^1 and N^2 respectively, with $N^1 \cap N^2 = \emptyset$ and $N^1 \cup N^2 = N$. Each ship $i \in N$ is characterised by its arrival time (or release date) r_i, width w_i, length l_i, draught d_i, travel direction and type. By assuming rectangular-shaped ships, we enable a straightforward evaluation of the placement constraints. The

representation of ships by rectangles is common practice as the exact shape of the ships is often not available to lock operators. The second major component of the problem is the lock. A lock consists of one or more independently operated chambers, which means that the operations of one chamber do not restrict its neighbouring chamber's operations in any way. Each chamber is of a specific type $t \in T$, defining the chamber's width W_t, length L_t, maximal draught D_t and minimal lockage duration P_t. Each chamber also has a front and back door, and a left and right quay, each of which are defined with respect to the ship's travel direction. The set of chambers of the same type is denoted by U_t.

A solution to the lock scheduling problem is given in the form of a series of lockage operations and their related lockages (lower part of Figure 2.4). A lockage operation contains the time table for moving the vessels into the lock, changing the water level, and enabling the ships to leave the chamber. The vessels that are transferred in said lockage operation are defined by the related lockage. A lockage defines the ships that are transferred together and their positioning in the chamber.

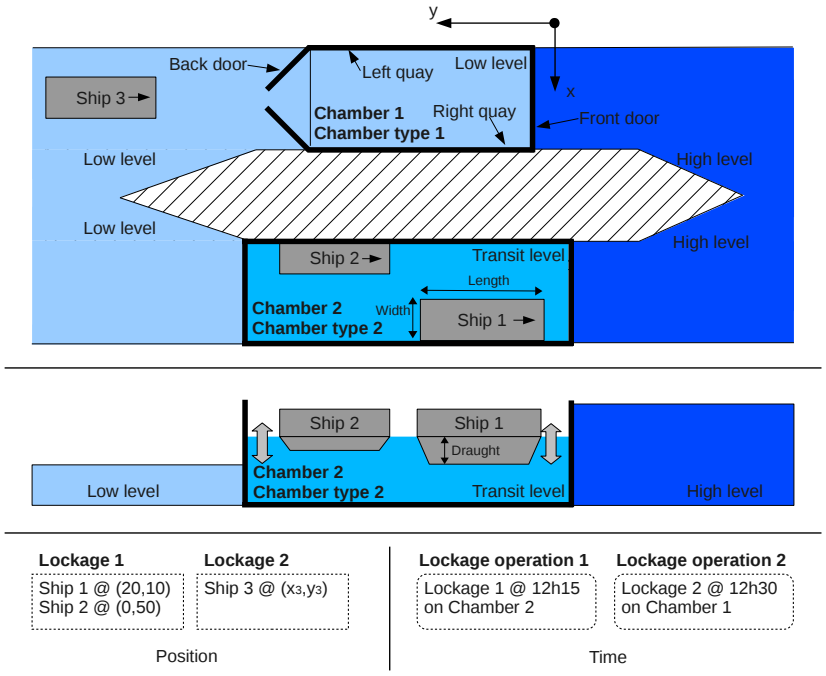

Figure 2.4: Representation of a multi chamber lock, and important lock scheduling terminology.

The LSP may have multiple objectives: minimise the total number of lockages, the completion time of the normal ships and/or the completion time of priority ships. A ship's completion time is defined as the time at which its associated lockage operation is finished, i.e. the chamber doors are completely open.

As aforementioned, the LSP encompasses three strongly interconnected sub problems: chamber assignment, lockage scheduling and ship placement.

Each ship is assigned a chamber type in the chamber assignment sub problem. The ship will thus be transferred in one of the lock's chambers of the assigned type. The constraining attributes are the ship's and the chamber type's dimensions: the ship's width, length and draught must be smaller than the limits for the chamber type.

The lockage scheduling sub problem can be modelled as a parallel machine scheduling problem where chambers map to machines and lockages to jobs. The sub problem also encompasses sequence dependent setup times, release dates, machine eligibility restrictions and time windows. The machines may have different processing speeds (lockage durations) as multiple chambers of different chamber types are allowed. This results in the following standard notation for the scheduling sub problem: $Q_m|r_k, s_{kl}, M_k|\sum w_k C_k$. The sequence dependent setup times are required for two reasons. Firstly, when two lockages in the same direction are processed directly after each other on the same chamber, an empty lockage in the opposite direction must be processed in between, resulting in a setup time between the two lockages. Secondly, when dealing with large ships approach and departure times must be considered for each vessel navigating the lock. Additionally, ships with tugboats require some time for (dis)engaging the tugboats before and after the actual lockage operation. A ship's pre-processing time is defined as the time required for the ship to approach the lock, moor to the quay or to another ship, disengage the tugboats (if any) and allow the tugboats to leave the lock. The post-processing time is the time required for the tugboats (if any) to enter the lock and engage the ship, and for the ship to unmoor and clear the lock. Both the approach time and departure time are defined in respect to a given 'coordination point' near the lock. Ships can overtake each other at these coordination points when necessary. Given that each lockage operation is feasible on one chamber type only machine eligibility restrictions are required. Otherwise a lockage defined for a large chamber type might be processed on a chamber of a much smaller type. The processing times can be ship dependent. For example, when a high number of barges are processed in a single lockage operation an additional processing time must be taken into account. Time windows correspond to the tidal windows for large sea vessels that can only approach or leave a port during high tide. Lastly, there is one additional constraint that enforces a *first-come-first-served* (FCFS) policy with respect to ship arrival times, either for all ships or for a subset.

This constraint defines that for two ships $i, j : r_i < r_j$, the lockage containing ship i must be finished no later than the lockage of ship j, i.e. $c_i \leq c_j$. Here we assume that no two ships ever arrive at the lock at exactly the same time.

The ship placement problem is subject to several constraints, visualised in Figures 2.5 and 2.6. The first set includes the standard 2D bin packing constraints, violated in Figures 2.5 (a) and (b). Rotating ships is also prohibited (Fig. 2.5 (c)). The other sub figures represent violations of the mooring constraints. When a ship is transferred in a lockage, it must be secured either to the right or left quay. This is achieved by mooring a ship directly to the quay or by mooring it to a larger ship that is already moored in a correct way (violated in Figure 2.5 (d), (e)). When mooring a smaller ship to a larger ship, the smaller ship must be contained within the length of the larger (Figure 2.5 (f)). When several types of vessels can be processed together, limitations may occur as to the vessel types a ship can moor to. In the case of mixed sea and inland traffic, for example, inland barges are normally unable to moor to seagoing vessels. Furthermore, in some locks ships cannot moor to each other when the difference in hull height above the water is too great (for example fully loaded and empty ships). These constraints can be viewed as group-mooring constraints that only allow mooring to ships that belong to a certain group. Figure 2.5 (g) shows how a barge may not moor to a seagoing vessel.

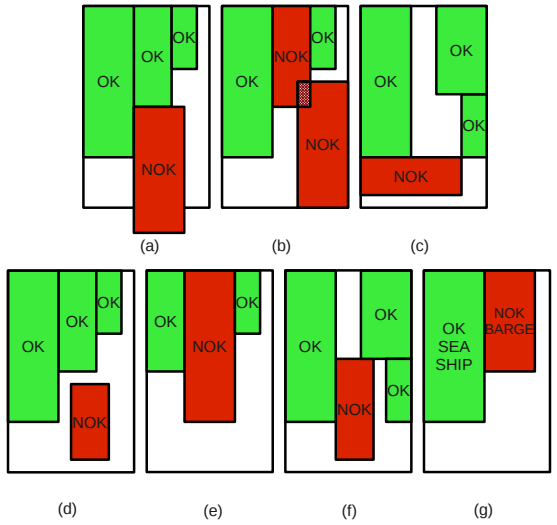

Figure 2.5: Representation of the hard constraints for the ship placement problem. (OK: ship placed in a correct way, NOK: ship placed incorrectly. Removing a NOK ship would make the packing feasible.)

The last set of additional constraints deals with the typical safety distances of the ship placement problem. Some manoeuvring space must be allocated to ships, preventing collisions while sailing in and out of the chamber, and in case of a possible minor accident during the lockage operation. While these safety distances can be considered equal for all ship pairs in an inland setting, traffic in sea ports requires a more individualised approach. The safety distance constraints for different ship pairs are visualised in Figure 2.6. Based on the dimensions and type of both ships, a minimal lateral and longitudinal safety distance can be calculated (Figure 2.6 (a)). When both ships use tugboats a sufficiently large lateral distance or 'corridor' must be present to enable the tugboats to sail between the ships when leaving the chamber before the lockage operation starts (Figure 2.6 (b)). The distance between each ship and the doors of the chamber (Figure 2.6 (c)) determines another set of constraints. These are required both for safety (i.e. to avoid collisions with the doors) and practical reasons (position of the mooring poles to which the ship can moor). This distance is dependent on both the chamber type and the ship's dimensions.

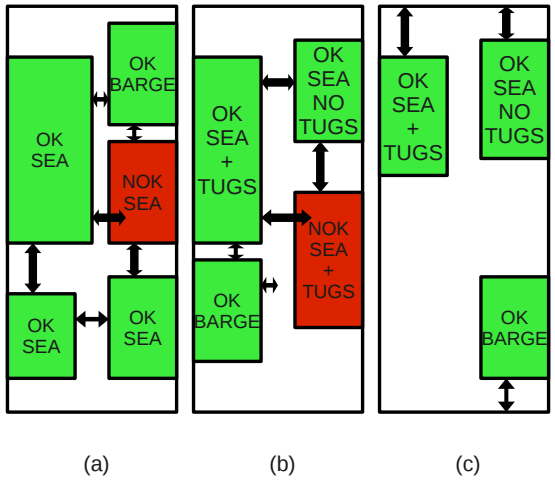

Figure 2.6: Representation of the lock scheduling problem's safety distance constraints. (OK: ship placed correctly, NOK: ship violates a safety distance constraint)

2.5 Mathematical model

This section introduces a *mixed integer linear programming model* (MILP) for the generalized lock scheduling problem. All variables are **emboldened**, whereas parameters retain regular formatting. Constraints belonging to the same sub problem of the LSP are grouped together and visualized, along with the constraints linking them to each other, in Figure 2.7.

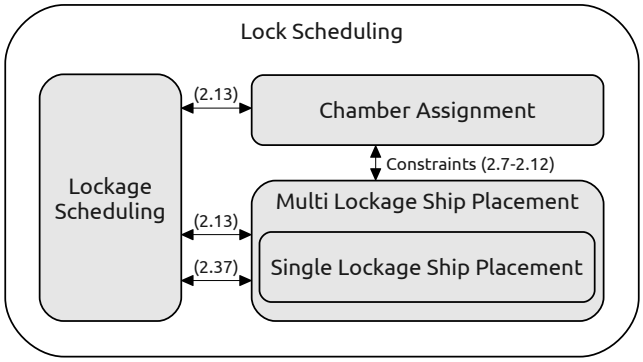

Figure 2.7: Representation of the mathematical model for the lock scheduling problem displaying its sub problems and the constraints linking these sub problems to each other.

The model contains additional ships that represent the left ($ship_0$) and right ($ship_{n+t}$) quays of each chamber of type $t \in T$. These 'quay' ships enable a straightforward implementation of the mooring constraints as the quays can now be seen as large ships with a fixed position to which a ship can be moored. Their functionality is explained below in Constraints (2.18), (2.19), (2.24) and (2.25), and Figure 2.9. The model also uses a set $MOOR_i$ containing all ships ship $i \in N$ is allowed to moor to. In the inland setting, this set contains all ships that are longer than ship i, as a ship can only moor to ships that are at least equally long. In a more general setting, this set may contain only ships of a specific ship type. Seagoing vessels, for example, may often moor only to the quay, while no other ships are allowed to moor to a seagoing vessel. By changing the ships that are available in the $MOOR_i$ sets, any such mooring restriction can be implemented. The model takes into account safety distances between ships, and between each ship and the chamber doors. These distances depend on ship properties such as dimensions, types and whether or not the ship requires tugboats. While the lateral safety distance between a seagoing vessel and a barge may average at 1.5 m, the minimal lateral distance between two seagoing vessels both requiring tugboats may be twelve meters.

The ship placement and chamber type assignment parts are inspired by the model for 2D bin packing with multiple bin sizes from Pisinger and Sigurd [2005]. Constraints (2.2) to (2.31) define the chamber assignment and ship placement parts of the problem. The scheduling part is based on the model for early/tardy scheduling with sequence dependent setup times on uniform parallel machines by Balakrishnan et al. [1999]. It is defined by constraints (2.32) to (2.44). Constraints (2.45) to (2.58) define the variables. All the parameters and variables are explained below.

Sets:

N, N^1, N^2:	$N = N^1 \cup N^2$ is the set of ships, with $\|N\| = n$, divided into upstream ships N^1 and downstream ships N^2.
$MOOR_i$:	Set of ships to which ship i can moor.
$TIDAL$:	Set of all ships with a tidal window.
$FCFS$:	Set of all ships that must be processed first-come-first-served with respect to their arrival times.
T:	Set of different chamber types.
U_t:	Set of chambers of type t, $t \in T$.
M:	$M = M^1 \cup M^2$ is the set of lockages, with $\|M\| = m$, thereby distinguishing between upstream (M^1) and downstream (M^2) lockages.
M_t:	$M_t = M_t^1 \cup M_t^2$ is the set of lockages suitable for chambers of type $t \in T$, again distinguishing between upstream and downstream lockages respectively. Note that M_t is an ordered set, i.e. $M_t = \{1, 2, \ldots, m_t^1, m_t^1+1, \ldots, m_t^1+m_t^2\}$, where m_t^i, $i = 1, 2$, are bounds on the number of upstream and downstream lockages for chamber type $t \in T$.

Parameters:

w_i, l_i, d_i:	Width, length and draught of ship i, $i \in N$ (integer).
r_i:	Time at which ship i, $i \in N$ arrives at the lock.
dF_{it}, dB_{it}:	Minimal distance between ship i, $i \in N$ and the front, back of a chamber of type t, $t \in T$.
sW_{ij}, sL_{ij}:	Minimal safety distance between ships i and j, $i, j \in N$ when they are lying next to, or behind each other.
$pre_i, post_i$:	Pre- and post-processing times of ship i, $i \in N$.
p_i:	Additional processing time induced by ship i, $i \in N$.
c_i^{min}, c_i^{max}:	Lower and upper limit on the completion time of ship i, $i \in N$.
wct_i:	Weight of the completion time of ship i, $i \in N$.
p_t:	The minimal processing time of a chamber of type t, $t \in T$.
$trans_{kl}$:	Transition time required between two consecutive lockages k, l, $k, l \in M_t$, $t \in T$ performed on the same chamber. $trans_{kl} = p_t$ if k, l are both upstream (or downstream) lockages, $trans_{kl} = 0$ otherwise.
W_t, L_t, D_t:	Width, length and draught of a chamber of type t, $t \in T$ (integer).
W, L, D:	Maximal width, length and draught over all chambers.

Variables:

$\boldsymbol{x_i, y_i}$:	Integer variables that define the x and y position of ship i, $i \in N$ (front left corner of the ship, Figure 2.8).
$\boldsymbol{left_{ij}}$:	Binary variable, $left_{ij} = 1 \Rightarrow$ ship i is completely to the left of ship j, $i, j \in N$.
$\boldsymbol{b_{ij}}$:	Binary variable, $b_{ij} = 1 \Rightarrow$ ship i is completely behind ship j, $i, j \in N$.
$\boldsymbol{ml_{ij}, mr_{ij}}$:	Binary variables, 1 when ship i is moored to ship j's left/right side, 0 otherwise, $i, j \in N$.
$\boldsymbol{z_k}$:	Binary variable, 1 when lockage k, $k \in M$, is used, 0 otherwise.
$\boldsymbol{f_{ik}}$:	Binary variable, 1 when ship i, $i \in N$, is processed in lockage k, $k \in M$, 0 otherwise.
$\boldsymbol{v_{ij}}$:	Binary variable, 1 when ship i and j, $i, j \in N$, are processed in the same lockage, 0 otherwise.
$\boldsymbol{c_i}$:	Departure time of ship i, $i \in N$ (completion time of the lockage).
$\boldsymbol{C_k}$	Completion time of lockage k, $k \in M$.
$\boldsymbol{P_k}$	Processing time of lockage k, $k \in M$.
$\boldsymbol{s_{kl}}$:	Setup time between lockages k and l, $k, l \in M$.
$\boldsymbol{seq_{kl}}$:	Binary variable, 1 when lockage k precedes lockage l, $k, l \in M$, in the same chamber, 0 otherwise.
$\boldsymbol{proc_{ku}}$:	Binary variable, 1 when lockage k, $k \in M_t$ is processed in chamber u, $u \in U_t$, $t \in T$, 0 otherwise.
$\boldsymbol{T_{max}}$:	Maximum lock transit time over all ships.

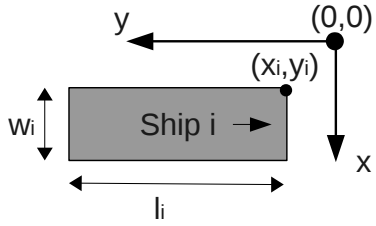

Figure 2.8: Clarification of the main axis and some important ship variables and parameters.

The objective (2.1) minimizes (1) the number of lockage operations, (2) the weighted completion times of the ships and (3) the maximum lock transit time, where $\lambda_1, \lambda_2, \lambda_3$ are independent weight factors. In case of a drought, for example, a greater weight can be given to the first part of the objective, ensuring the number of lockages used for transferring the ships is minimal. When a large queue of ships is waiting at the lock the weight of the maximum lock transit time can be increased so that none of the ships must wait a long amount of time.

$$min \;\; \lambda_1 \sum_{k} \boldsymbol{z_k} + \lambda_2 \sum_{i \in N} wct_i \boldsymbol{c_i} + \lambda_3 \boldsymbol{T_{max}} \tag{2.1}$$

The first block of constraints (2.2-2.31) models the ship placement part of the lock scheduling problem.
Constraints (2.2) to (2.4) ensure that two ships transferred in the same lockage do not overlap.

$$\boldsymbol{left_{ij}} + \boldsymbol{left_{ji}} + \boldsymbol{b_{ij}} + \boldsymbol{b_{ji}} + (1 - \boldsymbol{f_{ik}}) + (1 - \boldsymbol{f_{jk}}) \geq 1 \tag{2.2}$$

$$\forall i < j : \; i, j \in N_1 \vee i, j \in N_2, k \in M$$

$$\boldsymbol{x_i} - \boldsymbol{x_j} + W\boldsymbol{left_{ij}} \leq W - w_i, \quad \forall i, j : \; i, j \in N_1 \vee i, j \in N_2 \tag{2.3}$$

$$\boldsymbol{y_i} - \boldsymbol{y_j} + L\boldsymbol{b_{ij}} \leq L - l_i, \quad \forall i, j : \; i, j \in N_1 \vee i, j \in N_2 \tag{2.4}$$

The safety distance between two adjacent ships is modelled in Constraint (2.5) (Figure 2.6 (a)). The safety distance depends on both ships and their tugboat requirements. The minimal safety distance sW_{ij} between two vessels both requiring tugboats, for example, is larger than that between two ships where only one needs tugboats.

$$\boldsymbol{x_j} - \boldsymbol{x_i} + (W + sW_{ij})(1 - \boldsymbol{left_{ij}} + \boldsymbol{b_{ij}}) \geq w_i + sW_{ij}, \quad \forall i, j : \; i, j \in N_1 \vee i, j \in N_2 \tag{2.5}$$

When ship i is positioned behind ship j, constraint (2.6) ensures that safety distance requirements are met. This distance depends on the dimensions of both ships and on their ship types, see Figure 2.6 (b).

$$\boldsymbol{y_j} - \boldsymbol{y_i} + (L + sL_{ij})(1 - \boldsymbol{b_{ij}} + \boldsymbol{left_{ij}}) \geq l_i + sL_{ij}, \quad \forall i, j : \ i, j \in N_1 \vee i, j \in N_2 \tag{2.6}$$

Each ship must be placed within the dimensions of the chamber type in which it will be transferred. This restriction is modelled by Constraints (2.7) to (2.9). These constraints are examples of how the $\boldsymbol{f_{ik}}$ variables connect the $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ variables to a specific lockage.

$$\boldsymbol{x_i} + w_i \leq W_t + (1 - \boldsymbol{f_{ik}})W, \quad \forall i \in N, t \in T, k \in M_t \tag{2.7}$$

$$\boldsymbol{y_i} + l_i \leq L_t + (1 - \boldsymbol{f_{ik}})L, \quad \forall i \in N, t \in T, k \in M_t \tag{2.8}$$

$$d_i \leq D_t + (1 - \boldsymbol{f_{ik}})D, \quad \forall i \in N, t \in T, k \in M_t \tag{2.9}$$

Constraints (2.10) and (2.11) ensure the minimal safety distance between a ship and the front and back door of the chamber is maintained. These safety distances depend on the ship's type, its dimensions and the chamber used. The constraints are depicted in Figure 2.6 (c).

$$\boldsymbol{y_i} \geq dF_{it}(2\boldsymbol{f_{ik}} - 1), \quad \forall i \in N, t \in T, k \in M_t \tag{2.10}$$

$$\boldsymbol{y_i} + l_i \leq L_t - dB_{it} + (1 - \boldsymbol{f_{ik}})L, \quad \forall i \in N, t \in T, k \in M_t \tag{2.11}$$

Constraint (2.12) ensures each ship is transferred by exactly one lockage.

$$\sum_{k \in M} \boldsymbol{f_{ik}} = 1, \quad \forall i \in N \tag{2.12}$$

Constraint (2.13) models that each lockage transferring a ship must be executed.

$$\boldsymbol{f_{ik}} \leq \boldsymbol{z_k}, \quad \forall i \in N, k \in M \tag{2.13}$$

Constraints (2.14) to (2.31) describe the lock scheduling specific mooring constraints.

Constraints (2.14) and (2.15) model that ship i can only moor to ship j's right side when it is fully contained within ship j's side.

$$\boldsymbol{y_j} - \boldsymbol{y_i} \leq (1 - \boldsymbol{mr_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{2.14}$$

$$\boldsymbol{y_i} - \boldsymbol{y_j} \leq l_j - l_i + (1 - \boldsymbol{mr_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{2.15}$$

Only when ships i and j are adjacent can ship i be moored to ship j's right side. This is modelled by Constraints (2.16) and (2.17).

$$\boldsymbol{x_j} - \boldsymbol{x_i} \leq -w_j + (1 - \boldsymbol{mr_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{2.16}$$

$$\boldsymbol{x_j} - \boldsymbol{x_i} \geq -w_j - (1 - \boldsymbol{mr_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{2.17}$$

A ship can also moor to the right side of the left quay ($ship_0$) of the chamber by which it is transferred. Mooring a ship to the left quay is modelled by constraints (2.18) and (2.19), keeping in mind that $ship_0$ is the left quay for all chambers (see Figure 2.9).

$$x_0 - \boldsymbol{x_i} \leq (1 - \boldsymbol{mr_{i,0}})W, \quad \forall i \in N \tag{2.18}$$

$$x_0 - \boldsymbol{x_i} \geq (\boldsymbol{mr_{i,0}} - 1)W, \quad \forall i \in N \tag{2.19}$$

Constraints (2.20) - (2.23) model the constraints for mooring ship i to ship j's left side.

$$\boldsymbol{y_j} - \boldsymbol{y_i} \leq (1 - \boldsymbol{ml_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{2.20}$$

$$\boldsymbol{y_i} - \boldsymbol{y_j} \leq l_j - l_i + (1 - \boldsymbol{ml_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{2.21}$$

$$\boldsymbol{x_j} - \boldsymbol{x_i} \leq w_i + (1 - \boldsymbol{ml_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{2.22}$$

$$\boldsymbol{x_j} - \boldsymbol{x_i} \geq w_i - (1 - \boldsymbol{ml_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{2.23}$$

Mooring to the left side of the right quay ($ship_{n+t}$) of a chamber (of type t) is possible as well and modelled by Constraints (2.24) and (2.25). Constraint (2.26) ensures that a ship cannot be moored to the right quay of the wrong chamber type. When, for example, two chamber types are available at the lock, there will be two right quay ships ($n + 1$ and $n + 2$). A ship transferred in

chamber type 2 may moor to $ship_{n+2}$, but cannot moor to $ship_{n+1}$ (the right quay of chamber type 1). These constraints are represented in Figure 2.9.

$$x_{n+t} - \boldsymbol{x_i} \leq w_i + (1 - \boldsymbol{ml_{i,n+t}})W, \quad \forall i \in N, t \in T \tag{2.24}$$

$$x_{n+t} - \boldsymbol{x_i} \geq w_i - (1 - \boldsymbol{ml_{i,n+t}})W, \quad \forall i \in N, t \in T \tag{2.25}$$

$$\boldsymbol{ml_{i,n+t}} \leq \sum_{k \in M_t} \boldsymbol{f_{ik}}, \quad \forall i \in N, t \in T \tag{2.26}$$

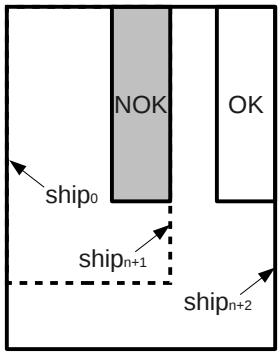

Figure 2.9: Visual representation of the constraints for mooring to the right quay of the different chamber types. The ship is transferred in a chamber of type 2 and is therefore unable to moor to the right quay of a chamber of type 1 ($ship_{n+1}$).

All ships have to be moored and this is modelled by Constraint (2.27). A ship can be moored to another ship, the left quay or one of the right quays.

$$\sum_{j \in MOOR_i} (\boldsymbol{ml_{ij}} + \boldsymbol{mr_{ij}}) + \boldsymbol{mr_{i,0}} + \sum_{t \in TYPES} \boldsymbol{ml_{i,n+t}} \geq 1, \quad \forall i \in N \tag{2.27}$$

When $ship_i$ is moored to $ship_j$, $ship_j$ cannot be moored to $ship_i$ and vice versa. This only occurs when both ships are equal in length, whereupon they could potentially moor to each other, thus rendering both ships unattached to the quay. Constraint (2.28) prevents this kind of 'fake' mooring.

$$\boldsymbol{ml_{ij}} + \boldsymbol{mr_{ji}} \leq 1, \quad \forall i \neq j,\ i,j :\ i,j \in N_1 \vee i,j \in N_2 \tag{2.28}$$

When $ship_i$ and $ship_j$ are in different lockages they cannot moor to each other. Constraints (2.29) to (2.31) ensure that the mooring constraints are only valid

for two ships that are transferred in the same lockage.

$$\boldsymbol{f_{ik}} - \boldsymbol{f_{jk}} \leq (1 - \boldsymbol{v_{ij}}), \quad \forall i < j : \ i, j \in N_1 \vee i, j \in N_2, k \in M \quad (2.29)$$

$$\boldsymbol{f_{jk}} - \boldsymbol{f_{ik}} \leq (1 - \boldsymbol{v_{ij}}), \quad \forall i < j : \ i, j \in N_1 \vee i, j \in N_2, k \in M \quad (2.30)$$

$$\boldsymbol{ml_{ij}} + \boldsymbol{mr_{ij}} + \boldsymbol{ml_{ji}} + \boldsymbol{mr_{ji}} \leq \boldsymbol{v_{ij}}, \quad \forall i < j, : \ i, j \in N_1 \vee i, j \in N_2 \quad (2.31)$$

The second block of constraints (2.32-2.44) models the scheduling part of the lock scheduling problem and is partially based on the model of Balakrishnan et al. [1999].

Constraints (2.32) and (2.33) ensure that the lockage completion time for $ship_i$ is equal to the completion time of the lockage in which it is transferred. C_{max} is a Big_M constant and should be sufficiently large.

$$\boldsymbol{c_i} \geq C_{max}(\boldsymbol{f_{ik}} - 1) + \boldsymbol{C_k}, \quad \forall i \in N, k \in M \quad (2.32)$$

$$\boldsymbol{c_i} \leq C_{max}(1 - \boldsymbol{f_{ik}}) + \boldsymbol{C_k}, \quad \forall i \in N, k \in M \quad (2.33)$$

When a ship is restricted by a tidal window, it must be processed before the tidal window ends. This is modelled by constraint (2.34).

$$c_i^{min} \leq \boldsymbol{c_i} \leq c_i^{max}, \quad \forall i \in TIDAL \quad (2.34)$$

Constraint (2.35) ensures that the processing time of lockage k is equal to the standard lockage duration for the lockage's chamber type t and is appropriately increased by the additional processing times of each ship that is transferred by that lockage.

$$\boldsymbol{P_k} \geq p_t \boldsymbol{z_k} + \sum_{i \in N} \boldsymbol{f_{ik}} p_i, \quad \forall k \in M_t, t \in TYPES \quad (2.35)$$

The setup time between lockage operations k and l is dependent on their direction and the ships that are processed. This is modelled in Constraint (2.36). When k and l process ships in the same direction the minimal setup time will be equal to p_t, the empty lockage operation time for the chamber type. This is the time required for changing the water level when the chamber is empty. If the lockage operations were in opposite directions the value would be zero, given that the water level need not be altered. The next part of the constraint states that both the post processing times of lockage operation k

and the pre-processing times of lockage operation l are accounted for.

$$\boldsymbol{s_{kl}} \geq trans_{kl} + \sum_{i \in N} \boldsymbol{f_{ik}} post_i + \sum_{i \in N} \boldsymbol{f_{il}} pre_i, \quad \forall l \neq k,\ k, l \in M_t, t \in T \tag{2.36}$$

Constraint (2.37) ensures that all activated lockages are assigned to one of the physical chambers corresponding to the lockage's chamber type. Lockages that that do not transfer any ships are not assigned to a physical chamber.

$$\sum_{u \in U_t} \boldsymbol{proc_{ku}} = \boldsymbol{z_k}, \quad \forall k \in M_t, t \in T \tag{2.37}$$

Constraint (2.38) ensures that two lockages can be sequenced after each other iff they are processed by the same physical chamber.

$$\boldsymbol{proc_{ku}} + \sum_{v \in U_t, v \neq u} \boldsymbol{proc_{lv}} + \boldsymbol{seq_{kl}} \leq 2, \quad \forall l > k,\ k, l \in M_t, u \in U_t, t \in T \tag{2.38}$$

The completion times of lockages that are processed by the same physical chamber are modelled using Constraints (2.39) and (2.40).

$$\boldsymbol{C_l} - \boldsymbol{C_k} + 2C_{max}(3 - \boldsymbol{seq_{kl}} - \boldsymbol{proc_{ku}} - \boldsymbol{proc_{lu}}) \geq \boldsymbol{P_l} + \boldsymbol{s_{kl}} \tag{2.39}$$
$$\forall k < l,\ k, l \in M_t, u \in U_t, t \in T$$

$$\boldsymbol{C_k} - \boldsymbol{C_l} + 2C_{max}(2 + \boldsymbol{seq_{kl}} - \boldsymbol{proc_{ku}} - \boldsymbol{proc_{lu}}) \geq \boldsymbol{P_k} + \boldsymbol{s_{lk}} \tag{2.40}$$
$$\forall k < l,\ k, l \in M_t, u \in U_t, t \in T$$

Lockage k cannot start before all ships in the lockage have arrived at the coordination point and have sailed into the chamber (Constraints 2.41).

$$\boldsymbol{C_k} - \boldsymbol{P_k} \geq \boldsymbol{f_{ik}}(r_i + pre_i), \quad \forall i \in N, k \in M \tag{2.41}$$

Constraint (2.42) ensures that a lockage is only scheduled when it transfers at least one ship.

$$\boldsymbol{z_k} \leq \sum_{i \in N} \boldsymbol{f_{ik}}, \quad \forall k \in M \tag{2.42}$$

The maximum lock transit time over all ships is defined by Constraint (2.43).

$$\boldsymbol{T_{max}} \geq \boldsymbol{c_i} - r_i, \quad \forall i \in N \tag{2.43}$$

Constraint (2.44) can be used to implement priority due to operational, economic or safety policies. It ensures that the transfer of $ship_j$ may not be completed before that of $ship_i$ when $ship_i$ and $ship_j$ have a first-come-first-served restriction ($i < j$).

$$\boldsymbol{c_i} \leq \boldsymbol{c_j}, \quad \forall i < j,\ i,j \in FCFS \tag{2.44}$$

Constraints (2.45-2.58) formulate bounds and integrality constraints on the variables.

$$\boldsymbol{left_{ij}},\ \boldsymbol{b_{ij}} \in \{0,1\}, \quad \forall i,j:\ i,j \in N_1 \vee i,j \in N_2 \tag{2.45}$$

$$\boldsymbol{ml_{ij}},\ \boldsymbol{mr_{ij}} \in \{0,1\}, \quad \forall i \in N, j \in MOOR_i \tag{2.46}$$

$$\boldsymbol{v_{ij}} \in \{0,1\}, \quad \forall i < j:\ i,j \in N_1 \vee i,j \in N_2 \tag{2.47}$$

$$0 \leq \boldsymbol{x_i} \leq W, \quad \forall i \in N \tag{2.48}$$

$$0 \leq \boldsymbol{y_i} \leq L, \quad \forall i \in N \tag{2.49}$$

$$0 \leq \boldsymbol{c_i} \leq C_{max}, \quad \forall i \in N \tag{2.50}$$

$$\boldsymbol{f_{ik}} \in \{0,1\}, \quad \forall i \in N, k \in M \tag{2.51}$$

$$0 \leq \boldsymbol{C_k} \leq C_{max}, \quad \forall k \in M \tag{2.52}$$

$$\boldsymbol{P_k} \geq 0, \quad \forall k \in M \tag{2.53}$$

$$\boldsymbol{z_k} \in \{0,1\}, \quad \forall k \in M \tag{2.54}$$

$$0 \leq \boldsymbol{s_{kl}} \leq C_{max}, \quad \forall k,l \in M \tag{2.55}$$

$$\boldsymbol{seq_{kl}} \in \{0,1\}, \quad \forall k,l \in M \tag{2.56}$$

$$\boldsymbol{proc_{ku}} \in \{0,1\}, \quad \forall k \in M_t, u \in U_t, t \in T \tag{2.57}$$

$$\boldsymbol{T_{max}} \geq 0 \tag{2.58}$$

A few assumptions were made with respect to safety distances and pre/post-processing times in the above model. Safety distances are independent of the chamber type and are defined only by the interaction between ships that are transferred together. Locks with chamber type-dependent safety distances between ships can, however, be tackled by the solution methods from Chapter 3 (for the SPP) and Chapter 5 (for the generalized LSP). Additionally, we

assume weather conditions are constant for the entire lock operation. This enables us to consider pre/post-processing times that depend only on the ships. When considering the pre/post-processing times, a distinction can be made between ships with and ships without tugboats. Ships with tugboats will always be positioned first in the chamber for any given lockage and their pre/post-processing times can be considered a constant value for a given weather condition. Ships without tugboats will be positioned later, and their pre/post-processing times may depend on the number of ships already in the chamber and on the remaining free space. The model also considers these much smaller times as constant, using averages based on the experience of lock masters.

Depending on the type of lock and on the traffic, several of the constraints in the model may be removed or reduced. When all ships have a draught that is smaller than the smallest chamber draught, Constraint (2.9) becomes obsolete. When there are no first-come-first-served limitations, Constraint (2.44) can be removed.

Another example of redundant constraints applies to inland locks. The processing times at these locks are not ship dependent, significantly reducing the impact of Constraints (2.35) and (2.36).

The number of ships in the $MOOR_i$ sets varies significantly between ship types. For ocean going vessels $MOOR_i$ will contain only the quay ships. Barges, on the other hand, may moor to any other barge, thus their $MOOR_i$ will also contain all barges that are at least as long as the barge itself.

When constructing the solution for a lock scheduling instance, the values of five variable groups must be retrieved from the MILP solution:

- The lockage in which each ship is transferred ($\boldsymbol{f_{ik}}$)
- The chamber used to process this lockage ($\boldsymbol{proc_{ku}}$)
- The ship's position in this chamber ($\boldsymbol{x_i}$, $\boldsymbol{y_i}$)
- The time at which the lockage operation ends ($\boldsymbol{C_k}$ or $\boldsymbol{c_i}$)
- The lockage operation duration ($\boldsymbol{P_k}$), enabling the calculation of the lockage start time

Given these variables' values, a lock scheduling solution can be constructed unambiguously.

Figure 2.10 shows how the main MILP variables are mapped to a solution for the lock scheduling problem. Each lockage contains at least one ship, each of which is placed at a specific location, with the x_i and y_i variables indicating the

location of their front left corner. An example of the interaction between ships in the same lockage is shown through the $mr_{9,5}$ and $v_{5,9}$ variables, which define how ship 9 is moored to ship 5, and that ships 5 and 9 are transferred together. The $f_{5,2}$ variable tells us that ship 5 is transferred in upstream lockage 2. When viewing the scheduling part, the figure visualizes how the $proc_{ku}$ variables map each lockage to a lockage operation conducted by a physical chamber. For example: lockage 1 is transferred in a lockage operation in physical chamber 2 ($proc_{1,2} = 1$) starting at 09:34 ($C_1 - P_1$).

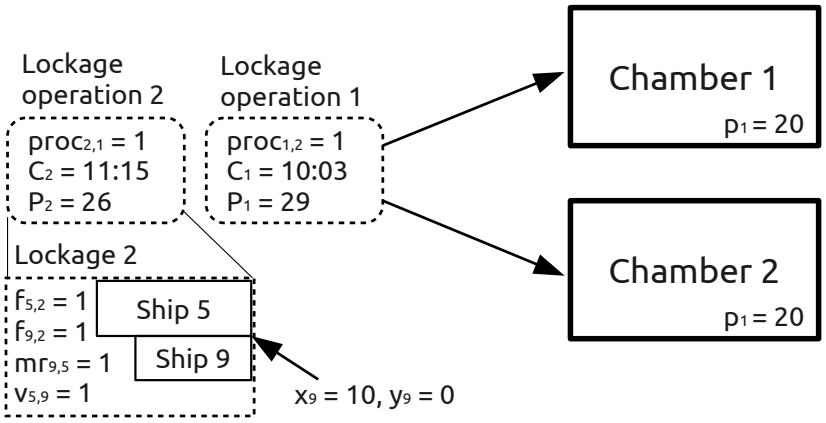

Figure 2.10: Representation of how the primary MILP variables are mapped to a solution for the lock scheduling problem.

2.6 Model speed ups

Several adaptations can be made to the model to reduce the size of the solution space and speed the solution process up. Firstly, the $\boldsymbol{z_k}$ and $\boldsymbol{v_{ij}}$ variables can be defined as continuous variables with values between 0 and 1. The nature of the model is such that omitting the zero-one constraint on the $\boldsymbol{z_k}$ and $\boldsymbol{v_{ij}}$ variables does not lead to incorrect fractional values in feasible solutions. Indeed, given that in a feasible solution the value of the $\boldsymbol{f_{ik}}$ variables is either 0 or 1, Constraints (2.13) and (2.42) force binary values for the $\boldsymbol{z_k}$ variables. Whenever Constraint (2.31) is active (otherwise $\boldsymbol{v_{ij}}$ is irrelevant) for a specific i-j combination, the binary values of $\boldsymbol{f_{ik}}$, $\boldsymbol{mr_{ij}}$ and $\boldsymbol{ml_{ij}}$ and Constraints (2.29)-(2.31) force a binary value for the corresponding $\boldsymbol{v_{ij}}$ variable. For a large set of instances that were solved in less than 12 hours, the introduction of these continuous variables reduces the computation time by an average of 33.75%.

A second optimization can be obtained by forcing an ordering in the lockages and lockage operation completion times. The first constraint (Constraint (2.59)) states that lockage l of type t can only be activated if lockage $k < l$ of the same type is already active. This symmetry breaking constraint strongly reduces computation time when the difference between the optimal solution and the lower bound is significant. The second constraint states that the completion time of lockage operation k of type t must be less than or equal to the completion time of lockage operation l of type t, where $k < l$ (Constraint (2.60)). Both restrictions are modelled through transitive constraints, which only put limitations on the subsequent lockage (operation) $k + 1$ of each lockage (operation) k. The addition of these transitive constraints decimated computation time on all tested instances, with a reduction of several orders of magnitude on some large instances.

$$\boldsymbol{z_{k+1}} \leq \boldsymbol{z_k}, \quad \forall k \in M_t, \forall t \in T \tag{2.59}$$

$$\boldsymbol{C_k} \leq \boldsymbol{C_{k+1}}, \quad \forall k \in M_t, \forall t \in T \tag{2.60}$$

The third improvement is obtained for instances where a FCFS policy is used with respect to the ship arrival times. Here we add an ordering constraint with respect to the lockage index of each ship, reducing the computation time with several orders of magnitude. When considering a single chamber lock, this constraint states that for any two ships i and j ($i < j$), the index (k) of the lockage that transfers ship i must be less than or equal to the index (l) of the lockage that transfers ship j (Constraint 2.61). In the case of multiple identical chambers this constraint is relaxed, requiring that $k \leq l + nrOfChambers$ (Constraint 2.62), thus allowing that a ship is transferred up to $|U| = nrOfChambers$ lockages before its predecessor. This relaxation is necessary as ships can be processed at the same time, but in different chambers. Figure 2.11 shows an example where the original Constraint (2.61) would exclude the, otherwise feasible, solution. Using the original constraint, ships 1 and 3 can be processed together if and only if ship 2 is transferred in the same lockage. As the presented solution does not violate the FCFS principle, this constraint should be replaced by constraint (2.62) to avoid excluding such feasible solutions. When multiple chamber types are available at the lock, the constraint is once again changed to ensure that the additional FCFS rule is only applied to ships that are processed in the same chamber type (Constraint (2.63)). By using the $FCFS$ sets instead of all the ships, these constraints can also be applied when only a subset of the ships are subject to FCFS constraints.

$$\sum_{k<c,\ k\in M} (\boldsymbol{f_{ik}} - \boldsymbol{f_{jk}}) \geq 0, \quad \forall i < j:\ i,j \in N_1 \vee i,j \in N_2, c \in M \tag{2.61}$$

$$\sum_{k<c+|U|,\ k\in M} \boldsymbol{f_{ik}} - \sum_{k=1}^{c} \boldsymbol{f_{jk}} \geq 0, \quad \forall i < j:\ i,j \in N_1 \vee i,j \in N_2, c \in M \tag{2.62}$$

$$-2 \sum_{k<c,k\in M_t} \boldsymbol{f_{jk}} - \sum_{k>c,\ k\in M_t} (\boldsymbol{f_{ik}} + \boldsymbol{f_{jk}}) + \sum_{c+1<k<c+|U_t|,\ k\in M_t} \boldsymbol{f_{ik}} \geq -2, \tag{2.63}$$

$$\forall i < j:\ i,j \in N_1 \vee i,j \in N_2, c \in M_t, t \in T$$

Figure 2.11: An example of a feasible solution for which the original index ordering constraint (2.61) does not hold.

2.7 Experiments

We have applied the mixed integer linear programming model for the LSP to several instances generated from historical data. These instances are split into two different sets, one for a single-traffic inland setting and one for locks that serve mixed traffic. For the inland setting, the influence of lock configurations (single/parallel small/large chamber(s) with/without FCFS) and ship inter arrival times on the ship waiting time and computation time will be analysed. A ship's waiting time is defined as the time between the completion time of its

lockage, and the earliest possible completion time of a lockage transferring the ship when no other traffic would be present at the lock. The influence of the increased complexity due to ship dependent safety distances on the computation time is analysed on the mixed-traffic instances, as well as the effect of dropping the FCFS policy. We assume that all ship arrival times are known beforehand. All experiments were run on an Intel® Core2Duo™E8400 cpu with 4GB memory running Windows XP SP3 and Gurobi 4.5.2 under an academic license. Gurobi was set to time-out after 12 hours and return the best solution found.

2.7.1 Single-traffic locks

The single-traffic test set is based on data from the waterway traffic on the Albertkanaal in Belgium. The locks on this canal have two identical small chambers and one large chamber, all of which can be operated independently. Table 2.1 presents the properties of these locks. The traffic at the locks was generated based on actual traffic from 2008 and the settings from Table 2.2. The instance properties are identifiable via the following convention: 'mean inter arrival time'-'number of ships'-'fraction of ships travelling upstream'. The ship sizes were extracted from 2008's traffic, which did not contain accurate information on ship arrivals. Therefore the ship sequence was determined randomly, and the inter arrival times were generated randomly between zero and two times the given mean inter arrival time. The data set is available online [Verstichel, 2012]. A first-come-first-served policy is used with respect to the ship arrival times to maintain fairness among the ships. This FCFS policy corresponds to the actual practice at the locks. Therefore, the additional constraints described in Section 2.6 were added to the model. We assume that all safety distances are included in the ship and chamber dimensions and that the number of ships in a lockage does not influence the processing time.

Table 2.1: Attributes of the locks on the Albertkanaal.

	Width (m)	Length (m)	p (min)	Chamber ID
Chamber type 1	16.0	136.0	16	1, 2
Chamber type 2	24.0	200.0	16	3

First, we compare the model's performance in four different lock configurations. In the *single small chamber* (SSC) setting only one chamber of type 1 is available at the lock. In the *single large chamber* (SLC) setting, one chamber of type 2 is available at the lock. The *parallel small chamber* (PSC) setting assumes a lock with two parallel chambers of type 1, while the *multi chamber type* (MCT) setting considers the actual lock from the Albertkanaal (Table 2.1). These

Table 2.2: Properties of the generated single-traffic instances.

Traffic properties	Instances
Mean inter arrival time (min)	5, 10, 15, 30
Number of ships	10, 20, 30
Upstream/Downstream traffic	50/50, 30/70
Safety distances	Included in ship dimensions
Example instance name	5-20-0.3

settings allow a thorough comparison of the computation and waiting times for different lock settings. The difference between using a single small and a single large chamber is also indicative of the influence of the ship placement part on calculation time. Table 2.3 shows an overview of the results for these four different inland test settings. The results were obtained under a weighted completion time objective ($\lambda_2 = 1.0$), with the minimization of the maximum waiting time as a secondary objective ($\lambda_3 = 0.1$). The total number of lockages was not taken into account for these experiments ($\lambda_1 = 0.0$). All single small chamber instances were solved to optimality in less than 20 seconds, while the large chamber instances required up to 336 seconds for attesting optimality. This suggests that the ship placement part of the lock scheduling problem strongly influences the hardness of an instance. While instances with parallel chambers took significantly more time for solving than the single small chamber cases, a reduction in the waiting time of at least 66.8% and a maximum of 100% was obtained. When comparing the waiting times for a single large chamber and the parallel small chambers, there are some interesting differences. Using the single large chamber is on average 14% faster than using the small parallel chambers for small ship inter arrival times (5 minutes). When the inter arrival time increases to 10, 15 or 30 minutes, the lock with parallel chambers reduces the average waiting time by 15%, 73% and 90% respectively compared to using the single large chamber. Regarding the MCT setting, the optimal solutions are found slower than in cases with a single chamber type, but the resulting reduction in waiting time remains significant. Careful examination of the search logs provided by Gurobi did however show that the optimal solution was found early in the search in almost all cases and that the remaining computation time was used to prove optimality.

A second set of experiments was performed on the smallest test instances for inland locks. We removed the first-come-first-served policy from the model for the SSC, PSC and MCT settings. As a consequence the lockage index constraints (2.61–2.63) could not be applied. The results for these experiments are presented in Table 2.4, from which it is clear that the FCFS constraints strongly reduce the calculation time. For the single chamber setting, the model

with FCFS constraints is between 43 and 293 times faster, while the waiting time increases between 0% en 27.4%. For parallel chambers, the FCFS model is between 1.5 and 159 times faster, while the total waiting time is increased for just one instance. When considering the real-life lock, both approaches are the fastest on 50% of the instances. In the first two lock configurations the number of explored branch&bound nodes became much larger after dropping the FCFS policy. For the SSC setting the average number of explored nodes increased from 5 to 903, while the PSC setting results in an average increase from 3071 to 15394 explored nodes. For the MCT setting the average number of explored nodes is 45177 and 3096 respectively. The efficiency of the FCFS lockage index constraint decreases with each relaxation from the single chamber (Constraint (2.61)) to the parallel chamber (Constraint (2.62)) and to the multi chamber type (Constraint (2.63)) version. Where the original single chamber constraint reduced the number of explored nodes by a factor of 180, the reduction drops to a factor 5 after the first relaxation. However, in both cases the effect on the calculation time is considerable. While the relaxation to the multi chamber type version of the constraint reduced the number of explored nodes by a factor 15, the computation time improvement is less pronounced than for the other two settings.

Table 2.3: Results for the single-traffic instances. '#' Shows the number of lockages, 'WT' the total waiting time in minutes and 'Calc' the calculation time in seconds.

	SSC			SLC			PSC			MCT		
Instance	#	WT	Calc	#	WT	Calc	#	WT	Calc	#	WT	Calc
5-10-0.3	6	218	0.48	4	110	2.75	6	66	2.33	6	21	41.73
5-10-0.5	7	292	0.11	4	74	2.73	7	52	2.28	8	5	28.02
10-10-0.3	10	467	0.09	6	107	0.90	9	115	1.14	9	11	33.63
10-10-0.5	8	252	0.08	5	103	1.49	9	54	1.26	9	15	30.64
15-10-0.3	9	148	0.09	7	64	1.65	10	2	0.28	10	0	25.69
15-10-0.5	7	182	0.13	7	81	2.41	9	17	0.33	10	0	25.63
30-10-0.3	8	15	0.2	8	15	1.71	10	0	0.11	10	0	24.45
30-10-0.5	10	48	0.64	10	48	1.71	10	0	0.11	10	0	26.34
5-20-0.3	15	1829	1.28	6	297	251.51	16	606	353.91	14	59	$\sim 3.9h$
5-20-0.5	13	1350	0.83	7	211	10.32	13	379	$\sim 2h$	13	37	$\sim 5h$
10-20-0.3	16	1352	1.08	8	203	11.16	16	241	177.03	18	17	1305.73
10-20-0.5	16	1025	0.59	8	163	9.34	17	130	478.62	17	16	3207.56
15-20-0.3	14	600	19.58	10	148	10.00	17	77	215.21	18	9	$\sim 6.2h$
15-20-0.5	16	863	1.27	13	171	18.65	20	7	4.42	20	0	29.72
30-20-0.3	19	679	1.19	12	117	10.12	19	33	5.05	20	2	$> 12h$
30-20-0.5	17	182	1.17	15	123	10.43	19	14	1.7	20	0	30.12
5-30-0.3	18	2034	6.64	8	350	157.74	19	422	$> 12h$	19	75	$> 12h$
5-30-0.5	15	1017	6.31	9	340	45.36	17	183	$> 12h$	20	52	$> 12h$
10-30-0.3	23	2407	11.78	13	188	56.65	26	136	$\sim 2.8h$	24	15	$> 12h$
10-30-0.5	22	1308	3.7	16	290	336.32	23	227	$> 12h$	28	13	$> 12h$
15-30-0.3	23	1410	4.36	17	218	123.51	26	111	$> 12h$	28	3	$> 12h$
15-30-0.5	24	795	4.59	18	197	135.54	26	57	$\sim 1h$	27	10	$> 12h$
30-30-0.3	26	119	6.11	26	98	121.34	29	12	89.39	29	2	454.58
30-30-0.5	27	114	3.66	26	82	48.89	29	7	40.83	28	1	83.47

Table 2.4: Results for the experiments without FCFS policy on the single-traffic instances. '#' Shows the number of lockages, 'WT' the total waiting time in minutes and 'Calc' the calculation time in seconds.

	SSC			PSC			MCT		
Instance	#	WT	Calc	#	WT	Calc	#	WT	Calc
5-10-0.3	6	218	23.39	6	66	34.34	5	21	65.61
5-10-0.5	7	258	27.23	7	52	4.44	8	5	30.19
10-10-0.3	9	339	27.16	9	81	23.33	9	11	1681.97
10-10-0.5	8	220	21.94	9	54	26.41	9	15	45.36
15-10-0.3	9	136	27.58	10	2	0.47	10	0	19.25
15-10-0.5	7	150	27.45	9	17	19.08	10	0	23.52
30-10-0.3	8	15	27.50	10	0	17.42	10	0	0.72
30-10-0.5	10	48	28.02	10	0	17.28	10	0	0.70

2.7.2 Mixed-traffic locks

The mixed-traffic test set is based on historical data from the port of Antwerp. The tide-independent docks of this port are connected to the river Scheldt through four different lock complexes. The presented test instances are based on historical data from the *Berendrecht-Zandvliet* (BE-ZV) lock complex and the *Van Cauwelaert* (VC) lock, with instance sizes ranging from 10 to 28 ships. The independent weight factors for the objective are the same as for the previous experiments: $\lambda_1 = 0.0$, $\lambda_2 = 1.0$, $\lambda_3 = 0.1$. The instances were provided to the authors under a non-disclosure agreement and can therefore not be published online. The properties of the locks are provided in Table 2.5.

The results, both with and without the first-come-first-served constraints, are presented in Table 2.6. These results show that ship dependent safety distances, setup times and processing times significantly increase the calculation time, even for the smallest instances, when compared with the single-traffic case. Adding a FCFS policy again strongly reduces the calculation time and has a limited impact on the weighted completion time objective. For one instance, VC-28, the FCFS solution is better than the solution of the original model. This can be explained by the slow convergence of the model without first-come-first-served limitations: the objective was still improving at the 12h time-out, so optimality was yet to be reached. Given this limited influence of the first-come-first-served policy on the solution quality and its large decrease in calculation time, its application is definitely worthwhile. Even when the FCFS policy is applied to only a subset of the ships, the resulting decrease in computation time may be considerable.

Table 2.5: Attributes of selected locks from the Port of Antwerp.

	Width (m)	Length (m)	p (min)	Chamber ID
Berendrecht	68.0	500.0	30	BE
Zandvliet	57.0	500.0	30	ZV
Van Cauwelaert	35.0	270.0	30	VC

Table 2.6: Experiment results for the mixed-traffic instances. 'Calc' shows the calculation time in seconds, 'Cost' the objective cost of the solution.

		No FCFS		FCFS	
Lock	#Ships	Cost	Calc	Cost	Calc
BE-ZV	10	**3076.5**	4917.96	3086.8	23.15
BE-ZV	12	**2493.7**	$> 12h$	2623.5	72.28
BE-ZV	13	**3611.8**	$> 12h$	3615.3	129.78
VC	10	**2920.2**	18.50	**2920.2**	17.40
VC	12	**5683.3**	20.37	**5683.3**	17.83
VC	23	**16522.3**	3906.45	**16522.3**	76.00
VC	28	32634.6	$> 12h$	**32589.5**	2628.35

2.8 Conclusion

Locks are an important part of the maritime infrastructure. They are used to transfer ships from one water level to another on inland waterways, between tide-dependent and tide-independent parts of ports and even between oceans. While some research has been published on lock scheduling, up until now a complete description and realistic model for single and parallel chamber locks transferring one or more ships in a single lockage operation was missing from academic literature. Furthermore, the ship placement problem, one of the LSP's sub problems, was either ignored or simplified to the well known two-dimensional rectangular bin packing.

When solving the lock scheduling problem, three strongly interconnected sub problems can be identified: chamber assignment, lockage scheduling and ship placement. A mixed integer linear programming model comprising all three sub problems was developed. The model represents real-life lock scheduling for both single and mixed-traffic locks, under a wide range of operational and economical constraints while using a weighted objective function. Through detailed modelling with great attention for real-life issues, this model generates optimal and real-life feasible solutions to small lock scheduling instances.

Furthermore, the model is very flexible, since its modular structure allows for easy addition and removal of constraints and objective terms, depending on the given situation. This includes specific settings for locks in ports or on inland waterways, but also environmental problems such as drought and operational issues like large queues. The model is further improved upon by adding constraints that reduce calculation time and tree size by several orders of magnitude when a first-come-first-served policy is applicable.

Experiments on instances generated from historical data demonstrate how some single-traffic instances with up to 30 ships can be solved to optimality in less than 12 hours for an inland setting, while mixed-traffic instances with only 12 ships could not be solved in less than half a day. These long calculation times for small to medium instances limit the practical applicability of the presented model. In practice, five to ten minutes of computation time is an absolute maximum for generating solutions, and calculation time should be predictable. Furthermore, the current approach does not guarantee any kind of convergence (and thus solution quality) in such short calculation times. Consequently, a solution generated in five minutes might easily be improved upon by a human expert. From the experimental point of view, the model's true value lies in its application as a reference for faster (heuristic) solution methods: while most exact solution methods for complex combinatorial problems solve a simplified version of the actual problem (thus actually providing a lower bound instead of an optimal solution) the presented model generates optimal solutions for the actual real-life generalized lock scheduling problem.

Chapter 3

The ship placement problem

The ship placement problem is the sub problem of the LSP that determines the position of the ships in the chambers during the lockage operation. These positions are determined by constraints (2.2) to (2.31) in the mathematical model for the lock scheduling problem (Section 2.5). Placing ships in a chamber can however also be considered separate from the chamber assignment and scheduling aspects of the LSP. Either as a stand-alone problem (Chapter 4), or as part of a (decomposition-based) solution method for the LSP (Chapter 5). In the latter case, some interaction between the SPP and the other sub problems must be possible, allowing the supervising method to alter the play field of the SPP, enabling the generation of different (possibly worse) ship placement solutions in its search towards a better LSP solution.

We start this chapter with a formal definition of the ship placement problem and the identification of similar 2-dimensional (2D) packing problems from literature in Section 3.1. We continue with a literature review of 2D rectangle packing problems in Section 3.2. A mathematical model is presented in Section 3.3. In Section 3.4 the first-in-first-out constraints of the ship placement problem are exploited to partition it into a sequence small single-lockage ship placement instances. An exact approach to this decomposed SPP is presented in Section 3.5, while a heuristic method is presented in Section 3.6. Section 3.7 contains a thorough computational study of the algorithms.

This chapter is based on Verstichel et al. [2013a] and Verstichel et al. [2013c].

3.1 Problem definition

Given an ordered list of n ships, the SPP aims at minimising the number of lockages needed to place all ships, subject to a number of specific placement and sequence constraints. The problem is reminiscent of the classic 2D bin packing problem (or 2D rectangular SBSBPP [Wäscher et al., 2007]) where a set of rectangular items (ships) needs to be positioned inside as few rectangular bins (lockages) as possible, and where rotation of the items is not allowed. However, there are a number of relevant differences. First, a sequence constraint stipulates that the ships need to be processed in a first-in-first-out (FIFO) way with respect to their position in the ship list. If we assume that the lockages are ordered by their index, and that the ship at position i in the ship list is positioned in lockage k, then the ships at position $j > i$ in this list are not allowed in any lockage with index $l < k$. Second, all ships should be placed in the first lockage they fit in. This means that if the optimal number of lockages is 5, and ship 7 can be placed in either lockage 3 or lockage 4, it should be placed in lockage 3. Although it may seem that these constraints are of no immediate relevance for the ship placement itself, they are vital when connecting the ship placement problem to the scheduling part of the LSP. The lockages generated by a ship placement algorithm can be used by scheduling algorithms for the generalized lock scheduling problem. If the obtained lockages are sub-optimal from a scheduling point-of-view, the scheduling algorithm can change the ship order or reduce the number of ships that are transferred in a specific lockage, effectively generating a new instance for the ship placement algorithm. By iterating between both solution methods, a globally better solution to the lock scheduling problem can be found.

The third difference is the set of mooring and safety distance constraints. The mooring constraints state that each ship must be moored either to the quay, or to another ship. Geometrically, ship i is said to be moored to ship j, when ship i is adjacent to ship j over its entire length. This constraint implies that each ship will be connected to the quay through larger ships, or through ships of equal size. It should be noted that, contrary to its type, the width of a ship does not affect the evaluation of the mooring constraint. The safety distance constraints ensure that some ‘room’ is given to ships, allowing for corrections to prevent collisions while manoeuvring in and out of the chamber, and in case of a possible minor accident during the lockage operations. The same type of constraints ensure that a ‘corridor’ remains free for tugboats. Section 2.4 presents an in-depth analysis of mooring and safety distance constraints.

In the absence of the mooring and safety constraints, the ship placement problem is NP-hard. This follows from a result in Leung et al. [1990] stating that the problem of determining whether a given set of squares fits in a given square is

NP-complete.

One might wonder whether the mooring constraint adds to the complexity, or in other words, whether instances exist for which the 2D rectangular SBSBPP requires fewer bins (lockages) than the ship placement problem. The answer to this question is positive, and we provide two instances with this property. The first instance is shown in Figure 3.1 where application of the ship placement constraints results in a solution with two lockages, while the 2D bin packing problem requires only one bin. The 8 ships have the following dimensions: 12x3, 10x4, 9x4, 7x6, 7x4, 6x3, 6x3 and 5x2, while the chamber has dimension 19x13. Due to the small number of ships, the MILP model can be solved to optimality in less than 100 seconds. The obtained solution confirms that two lockages are required to place all ships when the ship placement constraints are taken into account. The second instance considers a perfect packing where the items have to be placed in three rows to obtain an optimal single bin solution. All 14 rectangles have a width of 1, and their lengths are 27 (2 items), 12 (5 items), 10 (6 items) and 6 (1 item), while the chamber has a dimension of 60x3. There exist six different optimal solutions for the 2D bin packing problem for the given instance, each of which violates at least one ship placement constraint. It follows that when this instance is viewed as a ship placement problem, at least two lockages are required. Figure 3.2 visualises these single bin solutions and highlights items that are placed in violation with the ship placement constraints. For simplicity's sake, the group mooring and safety constraints are omitted in these examples. When taken into consideration, they would lead to larger differences between the bin packing and ship placement solutions.

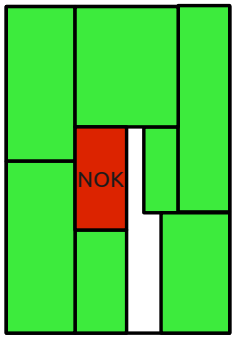

Figure 3.1: Example of an infeasible single lockage ship placement problem. (NOK: ship in violation with the ship placement constraints)

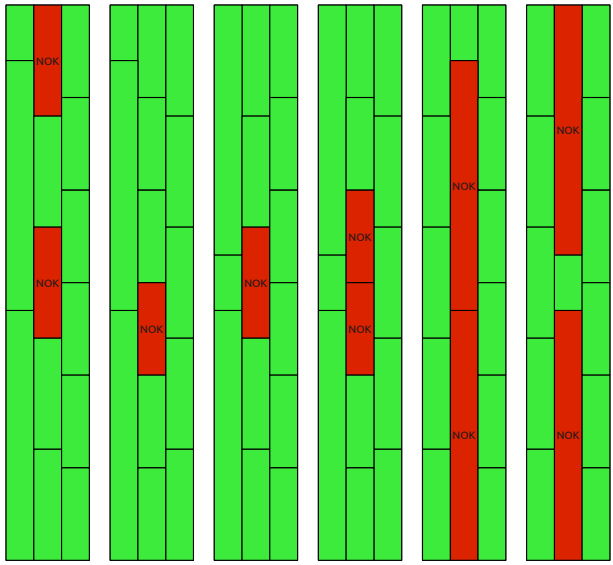

Figure 3.2: The six optimal 2D bin packing solutions for this problem all violate at least one ship placement constraint. (NOK: ship in violation with the ship placement constraints)

3.2 Literature review

The ship placement problem has not been addressed in the academic literature. Some papers do mention positioning ships inside lock chambers at the Three Gorges dam [Wang et al., 2013; Zhang et al., 2008], but they consider it identical to 2D bin packing. Contrastingly, numerous papers were published on the related two-dimensional packing problem, for which Dyckhoff [1990] and Wäscher et al. [2007] developed a topology. One of the proposed solution methods for the ship placement problem is derived from the best fit heuristic, which is an orthogonal strip packing heuristic. Therefore, the overview in this dissertation focusses on said problem. The aim of the orthogonal strip packing problem is to place a number of rectangular items on an infinitely tall rectangular sheet with a fixed width, without any of the items overlapping, resulting in the minimum sheet height requirement for placing every item. This problem proves to be NP-hard [Garey and Johnson, 1979]. The sheet is infinitely long, and therefore all items can be placed on one single sheet. 90 degree rotations of the items is allowed, so without loss of generality it can be assumed that the width (horizontal dimension) of a non-rotated item constitutes its largest dimension. This is called the item's default configuration, whereas an item rotated by 90 degrees is

labelled the rotated configuration. Several approaches have been established to this problem. Heuristic approaches, such as the bottom left (fill) heuristic and its variants are applied by Baker et al. [1980]; Chazelle [1983]; Jakobs [1996]. Burke et al. [2004] present a best-fit heuristic outperforming the bottom left based heuristics on all benchmarks with more than 50 items and the majority of smaller instances. Imahori and Yagiura [2010] reduce the time complexity of the best-fit heuristic to an optimal $O(n \log n)$ and show that the heuristic performs very well for very large instances. Aşık and Özcan [2009] also improve the results of the best-fit heuristic by considering the height of the gap and by introducing a more complex rectangle selection procedure. The improvement, however, comes at considerable computational cost.

Metaheuristic approaches to the orthogonal strip packing problem are often hybridisations that generate different input sequences for existing heuristic approaches in order to improve results. Hopper and Turton [2001] present an interesting comparison of metaheuristic approaches and genetic algorithms. A metaheuristic combining the best-fit heuristic and a simulated annealing bottom left fill hybridisation [Burke et al., 2009] further improves the results of Burke et al. [2004]. Alvarez-Valdes et al. [2008] introduce a GRASP based heuristic to solve the orthogonal strip packing problem with fixed orientation. Although their approach does not allow for rotation of the rectangles, the GRASP heuristic outperforms the other, rotating, metaheuristics on the orthogonal strip packing problem, while its computation time is comparable. Both the simulated annealing hybridisation and the GRASP approach have a time limit of 60 seconds in which to produce their results.

A mixed integer linear programming model for the ship placement problem is formulated in the following section. The applicability of this exact solution method for the NP-hard SPP is, however, limited: the computation times for realistic instance sizes are too long and unpredictable for application in the aforementioned settings, where short and predictable computation times are paramount. The best-fit heuristic combines these requirements with a high solution quality in a simple, flexible and easily extendible constructive heuristic and is therefore transformed into a ship placement algorithm in Section 3.6.

3.3 A mathematical model for the ship placement problem

By isolating the ship placement constraints from the LSP model presented in Section 2.5, a mathematical model for the ship placement problem can be formulated. A separate SPP can be solved for each chamber type, and therefore

several constraints can be simplified or removed altogether. In the remainder of this section, a multi-lockage or 'full' model for the SPP is presented. The model below is the single direction, single chamber type version of the ship placement part (Constraints (2.2) to (2.31)) of the mathematical model for the LSP from Chapter 2. It is added because some constraints could be simplified and several indexes removed. Figure 3.3 displays this model's position in the mathematical model for the LSP.

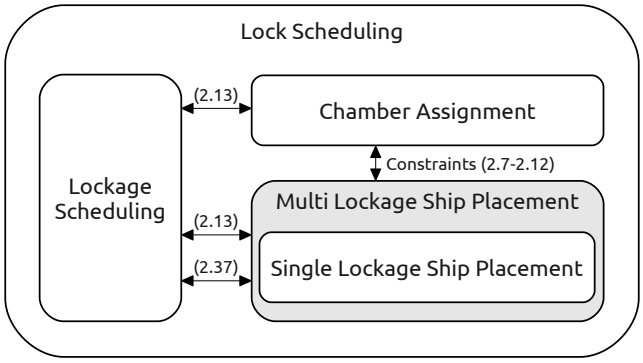

Figure 3.3: Representation of the mathematical model for the multi-lockage ship placement problem in the model for the lock scheduling problem.

The objective (3.1) is to minimize the number of lockages required for placing all the ships ($\sum_{k \in M} \boldsymbol{z_k}$). Among different solutions with an equal number of lockages, the one where all ships are placed in the lockage with the lowest possible index is favoured ($\sum_{k \in M} k \sum_{i \in N} \boldsymbol{f_{ik}}$). In other words, when $ship_i$ can be placed in either $lockage_k$ or $lockage_{k+1}$, while the resulting solutions have the same total number of lockages, it should be placed in $lockage_k$. To ensure that the total number of lockages is more important than the lockage in which a ship is placed, the main objective is multiplied by λ, which should be sufficiently large (e.g. $\lambda = |M|^2$).

$$minimize \;\; \lambda \sum_{k \in M} \boldsymbol{z_k} + \sum_{k \in M} k \sum_{i \in N} \boldsymbol{f_{ik}} \tag{3.1}$$

Constraints (3.2) to (3.4) ensure that two ships transferred in the same lockage do not overlap.

$$\boldsymbol{left_{ij}} + \boldsymbol{left_{ji}} + \boldsymbol{b_{ij}} + \boldsymbol{b_{ji}} + (1 - \boldsymbol{f_{ik}}) + (1 - \boldsymbol{f_{jk}}) \geq 1, \quad \forall i < j,\ i,j \in N, k \in M \tag{3.2}$$

$$\boldsymbol{x_i} + w_i \leq \boldsymbol{x_j} + W(1 - \boldsymbol{left_{ij}}), \quad \forall i \neq j,\ i,j \in N \tag{3.3}$$

$$\boldsymbol{y_i} + l_i \leq \boldsymbol{y_j} + L(1 - \boldsymbol{b_{ij}}), \quad \forall i \neq j,\ i,j \in N \tag{3.4}$$

The SPP is solved for a single chamber type at a time, therefore Constraints (3.5) and (3.6) suffice to ensure that all ships are placed inside the chamber and that the safety distances between ships and doors are respected.

$$\boldsymbol{x_i} \in \{0, \ldots, W - w_i\}, \quad \forall i \in N \tag{3.5}$$

$$\boldsymbol{y_i} \in \{dF_i, \ldots, L - l_i - dB_i\}, \quad \forall i \in N \tag{3.6}$$

Safety distances between ships are modelled in Constraints (3.7) and (3.8).

$$\boldsymbol{x_j} + (W + sW_{ij})(1 - \boldsymbol{left_{ij}} + \boldsymbol{b_{ij}}) \geq \boldsymbol{x_i} + w_i + sW_{ij}, \quad \forall i \neq j,\ i,j \in N \tag{3.7}$$

$$\boldsymbol{y_j} + (L + sL_{ij})(1 - \boldsymbol{b_{ij}} + \boldsymbol{left_{ij}}) \geq \boldsymbol{y_i} + l_i + sL_{ij}, \quad \forall i \neq j,\ i,j \in N \tag{3.8}$$

Constraint (3.9) ensures that each ship is transferred by exactly one lockage, while Constraint (3.10) models that each lockage transferring a ship must be executed.

$$\sum_{k \in M} \boldsymbol{f_{ik}} = 1, \quad \forall i \in N \tag{3.9}$$

$$\boldsymbol{f_{ik}} \leq \boldsymbol{z_k}, \quad \forall i \in N, k \in M \tag{3.10}$$

Ship i can only moor to ship j's right side when it is fully contained within ship j's side and when both ships are adjacent. This is modelled by Constraints

(3.11), (3.12), (3.13) and (3.14).

$$\boldsymbol{y_j} \leq \boldsymbol{y_i}(1 - \boldsymbol{mr_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{3.11}$$

$$\boldsymbol{y_i} + l_i \geq \boldsymbol{y_j} + l_j + (1 - \boldsymbol{mr_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{3.12}$$

$$\boldsymbol{x_j} + w_j \leq \boldsymbol{x_i} + (1 - \boldsymbol{mr_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{3.13}$$

$$\boldsymbol{x_j} + w_j \geq \boldsymbol{x_i} - (1 - \boldsymbol{mr_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{3.14}$$

A ship can also moor to the right side of the left quay (ship 0). Here it suffices to check that ships i and 0 are adjacent (Constraints (3.15) and (3.16)) because ship i will always be contained within the quay's length.

$$x_0 \leq \boldsymbol{x_i} + (1 - \boldsymbol{mr_{i,0}})W, \quad \forall i \in N \tag{3.15}$$

$$x_0 \geq \boldsymbol{x_i} - (1 - \boldsymbol{mr_{i,0}})W, \quad \forall i \in N \tag{3.16}$$

Ship i can only moor to ship j's left side when it is fully contained within ship j's side and when both ships are adjacent. This is modelled by Constraints (3.17), (3.18), (3.19) and (3.20).

$$\boldsymbol{y_j} \leq \boldsymbol{y_i} + (1 - \boldsymbol{ml_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{3.17}$$

$$\boldsymbol{y_i} + l_i \leq \boldsymbol{y_j} + l_j + (1 - \boldsymbol{ml_{ij}})L, \quad \forall i \in N, j \in MOOR_i \tag{3.18}$$

$$\boldsymbol{x_j} \leq \boldsymbol{x_i} + w_i + (1 - \boldsymbol{ml_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{3.19}$$

$$\boldsymbol{x_j} \geq \boldsymbol{x_i} + w_i - (1 - \boldsymbol{ml_{ij}})W, \quad \forall i \in N, j \in MOOR_i \tag{3.20}$$

A ship can also moor to the left side of the right quay (ship $n + 1$). Here it suffices to check that ships i and $n + 1$ are adjacent (Constraints (3.21) and (3.22)) because ship i will always be contained within the quay's length.

$$x_{n+t} \leq \boldsymbol{x_i} + w_i + (1 - \boldsymbol{ml_{i,n+1}})W, \quad \forall i \in N \tag{3.21}$$

$$x_{n+t} \geq \boldsymbol{x_i} + w_i - (1 - \boldsymbol{ml_{i,n+1}})W, \quad \forall i \in N \tag{3.22}$$

All ships have to be moored which is modelled by Constraint (3.23). A ship can be moored to another ship, the left quay or the right quay.

$$\sum_{j \neq i,\ j \in N} (\boldsymbol{ml_{ij}} + \boldsymbol{mr_{ij}}) + \boldsymbol{mr_{i,0}} + \boldsymbol{ml_{i,n+1}} \geq 1, \quad \forall i \in N \tag{3.23}$$

When $ship_i$ is moored to $ship_j$, $ship_j$ cannot be moored to $ship_i$ and vice versa. This only occurs when both ships are equal in length, whereupon they could potentially moor to each other, thus rendering both ships unattached to the quay. Constraint (3.24) prevents this kind of 'fake' mooring.

$$\boldsymbol{ml_{ij}} + \boldsymbol{mr_{ji}} \leq 1, \quad \forall i \neq j,\ i,j \in N \tag{3.24}$$

Ships transferred in different lockages cannot moor to each other. Constraints (3.25) to (3.27) ensure that the mooring constraints are only valid for two ships that are transferred in the same lockage.

$$\boldsymbol{f_{ik}} - \boldsymbol{f_{jk}} \leq \boldsymbol{v_{ij}}, \quad \forall i < j, i,j \in N, k \in M \tag{3.25}$$

$$\boldsymbol{f_{jk}} - \boldsymbol{f_{ik}} \leq \boldsymbol{v_{ij}}, \quad \forall i < j,\ i,j \in N, k \in M \tag{3.26}$$

$$\boldsymbol{ml_{ij}} + \boldsymbol{mr_{ij}} + \boldsymbol{ml_{ji}} + \boldsymbol{mr_{ji}} \leq (1 - \boldsymbol{v_{ij}}), \quad \forall i < j,\ i,j \in N \tag{3.27}$$

A first-in-first-out policy with respect to the ship indices is enforced by Constraint (3.28). This constraint makes sure that ship $j > i$ cannot be placed in a lockage $l < k$ when ship i is transferred in lockage k.

$$\sum_{k<c,\ k \in M} (\boldsymbol{f_{ik}} - \boldsymbol{f_{jk}}) \geq 0, \quad \forall i < j,\ i,j \in N, c \in M \tag{3.28}$$

Constraints (3.29) to (3.33) formulate bounds and integrality constraints on the variables.

$$\boldsymbol{left_{ij}},\ \boldsymbol{b_{ij}} \in \{0,1\}, \quad \forall i \neq j,\ i,j \in N \tag{3.29}$$

$$\boldsymbol{ml_{ij}},\ \boldsymbol{mr_{ij}} \in \{0,1\}, \quad \forall i \in N, j \in MOOR_i \tag{3.30}$$

$$\boldsymbol{v_{ij}} \in \{0,1\}, \quad \forall i < j,\ i,j \in N \tag{3.31}$$

$$\boldsymbol{z_k} \in \{0,1\}, \quad \forall k \in M \tag{3.32}$$

$$\boldsymbol{f_{ik}} \in \{0,1\}, \quad \forall i \in N, k \in M \tag{3.33}$$

3.4 A decomposition method for the ship placement problem

Solving the MILP model for the multi lockage ship placement problem is not the most efficient way of tackling the SPP. In fact, producing optimal solutions for the SPP with this model becomes very time consuming when the problem size increases. Fortunately, the SPP can be decomposed into a sequence of single-lockage ship placement instances by exploiting the first-in-first-out constraints. By constructing lockages one ship at a time and closing each lockage only when it is full, any SPP instance can be solved as a series of small to medium sized single-lockage instances. The solution method starts with a single empty lockage. Ships are added to this lockage one at a time in the order in which they appear in the ship list until no feasible solution can be found. At this point, the lockage is considered full, and the last feasible solution is stored. The algorithm now continues with the remaining ships. The pseudo code of the decomposition method is presented in Algorithm 1, in which the ship placement algorithms, either exact or heuristic, are called at line 6.

The advantage of applying this FIFO based partitioning method is threefold. Optimality is guaranteed when using an exact placement algorithm. By solving a single-lockage SPP, the mathematical model becomes much tighter than the multi lockage variant: a significant number of Big_M constraints and a multitude of binary variables can be removed. In addition, the decomposition method generates small to medium sized sub problems (The number of ships that can fit in a single lockage is seldom more than 18 in real lock settings).

Algorithm 1 Pseudo code of the SPP decomposition method.

```
Input: ShipList
 1: LockageList ← empty list
 2: Lockage ← new Lockage
 3: while not all ships placed do
 4:    if Placement feasible then
 5:       Lockage ← Placement
 6:       add next ship of ShipList to Placement
 7:    else
 8:       add Lockage to LockageList
 9:       Clear Placement
10:    end if
11: end while
12: return LockageList
```

3.5 An exact approach for the single-lockage ship placement problem

A compact mathematical model for the single-lockage SPP can be formulated based on the decomposition method introduced in the previous section. Most of the constraints, parameters and variables in this model are the same as in the model of Section 3.3. In the model below, only the altered constraints are discussed. For the unaltered ones, we refer to the aforementioned model. The objective is optional, as the only requirement is to find a feasible solution for the current number of ships. The position of this model in the model for the LSP is visualized in Figure 3.4.

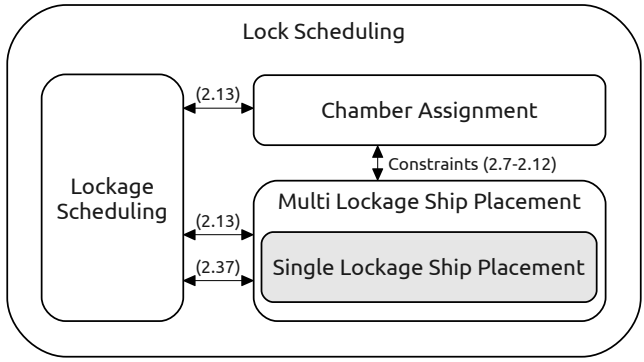

Figure 3.4: Representation of the mathematical model for the single-lockage ship placement problem in the model for the lock scheduling problem.

$$minimize \sum_{i=1}^{N} l_i \, \boldsymbol{y_i} \tag{3.34}$$

s.t.

$$\boldsymbol{left_{ij}} + \boldsymbol{left_{ji}} + \boldsymbol{b_{ij}} + \boldsymbol{b_{ji}} \geq 1, \quad \forall i < j, \; i,j \in N \tag{3.35}$$

(3.3), (3.4), (3.5), (3.6), (3.7), (3.8), (3.11), (3.12), (3.13), (3.14), (3.15), (3.16), (3.17), (3.18), (3.19), (3.20), (3.21), (3.22), (3.23), (3.24)

$$\boldsymbol{x_i} \leq \boldsymbol{x_j}, \quad \forall i < j, \; i,j \in N : w_i = w_j, l_i = l_j \tag{3.36}$$

Constraint (3.35) guarantees that for each ship pair in the lockage, one is left of and/or behind the other and corresponds to the single chamber version of

Constraint (3.2). Constraint (3.36) is new, and breaks some of the symmetry that is introduced when the problem deals with ships of the same size, by imposing an order in their x-position based on their position in the ship list. Thus, if two ships i and $j > i$ are identical, $\boldsymbol{x_i}$ will never be larger than $\boldsymbol{x_j}$, and a solution where ship i and ship j swap places will not be considered when searching for/proving the optimal solution. Adding this constraint results in a considerable speedup of the solution generation when an objective is taken into account, as opposed to returning the first feasible result.

The combination of the mooring constraints and the integer widths and lengths forces the x and y variables to be integer in any feasible solution. As a result, they can be defined as real variables, with bounds ensuring that they are placed inside the chamber's dimensions.

An additional speedup can be achieved by applying a lower bound heuristic. The current single-lockage ship placement instance is first solved by some heuristic. If this heuristic is able to position all ships in the chamber, its solution is returned. When the heuristic fails, the exact method is applied guaranteeing that the solution returned will be optimal. Applying a fast and effective heuristic such as the multi-order best-fit heuristic (Section 3.6), enables generating optimal solutions faster than solving the mathematical model at each step.

The mathematical model can be considered a feasibility problem, which can be solved faster than the corresponding optimization problem. There is no difference between the solutions of the feasibility problem and the optimization problem, apart from the position of the ships in the chamber. These actual positions do not influence the solution quality. Nevertheless, considering an objective such as placing the longest ships front-most in the chamber (3.34) may produce solutions that have some features in common with the results that are currently produced by human planners. Examples of such objectives are: positioning the first arriving ships at the front-most positions, mooring as many ships as possible to the left quay, etc.

3.6 A multi-order best-fit heuristic for the single-lockage ship placement problem

The multi-order best-fit heuristic for the single-lockage ship placement problem is based on the three-way best-fit heuristic [Verstichel et al., 2013c], which is an extension of the best-fit heuristic for the orthogonal strip packing problem [Burke et al., 2004]. Throughout the section which follows hereunder, the multi-order best-fit heuristic is presented. Only those aspects of the three-way

best-fit heuristics that are relevant for the ship placement problem are discussed in this section. A complete and detailed discussion of the three-way best-fit heuristic (including item rotation, optimal time behaviour and a comparison to other methods) is included in Appendix A which summarizes Verstichel et al. [2013c].

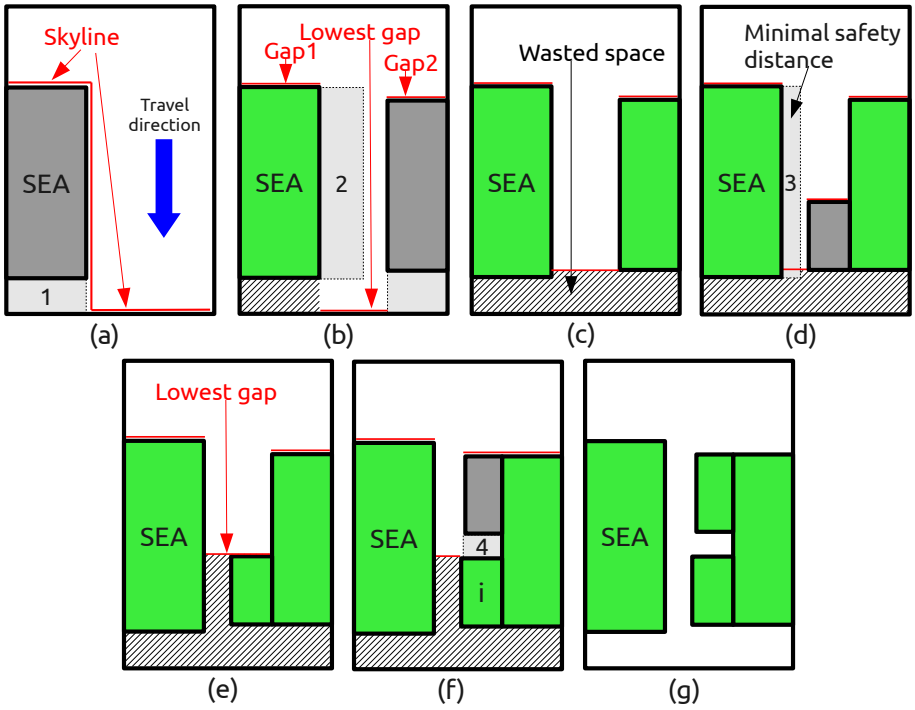

Figure 3.5: Representation of the gaps, based on a single best-fit iteration with a decreasing width ordering and a leftmost placement strategy. Newly placed ships are dark grey, safety distances are light grey with dashed lines, gaps are red, and wasted space is hatched.

The multi-order best-fit heuristic is a constructive heuristic that generates a limited number of solutions while searching for a feasible ship placement solution. It consists of three core components: a ship ordering method, a ship placement policy and an array of gaps. The ordering method determines the priority with which each ship will be placed. Any ordering can be applied, but a good balance between computation time and solution quality can be obtained by applying a few orderings, where each one introduces a significant disruption in the ship sequence of the other orderings. The placement policy determines the preferred

mooring side (left or right). The gaps form a skyline determining the free space in the chamber, thereby removing the need for costly overlap checks. Figure 3.5 illustrates the attributes of a gap. The figure visualizes a single best-fit solution construction, using a decreasing width ordering and a leftmost placement policy. Aside from the gap's width, its left and right neighbouring ships also determine whether or not a ship can 'fill' it: the candidate ship must be allowed to moor to either the left or the right neighbouring ship. Furthermore, the new ship must respect the minimum lateral safety distance ('2' and '3') between itself and its neighbours (Figure 3.5 (b) and (d) respectively). Each gap also has some so-called bottom allocations that determine the minimal y-position of the newly placed ship. The front door denotes the bottom allocation in Figure 3.5 (a), and a minimal safety distance '1' is required. Ship i forms the bottom allocation of the lowest gap in Figure 3.5 (f), inducing safety distance '4' between ship i and the newly placed ship. These neighbouring ships and bottom allocations help determining the safety distances that must be accounted for when positioning a ship in a gap.

A pseudo code of the multi-order best-fit heuristic for the ship placement problem is presented in Algorithm 2. The pseudo code refers to corresponding parts of Figure 3.6. In what follows, this heuristic is informally described, using three orderings: decreasing width, decreasing length and decreasing surface, and the leftmost placement policy. Other orderings such as increasing arrival time, can also be included in the multi-order best-fit heuristic.

Similar to the exact approach, ships are added to the chamber one by one until no more ships can be added without violating constraints. The set of ships currently under consideration for placement in the chamber is ordered by decreasing width. The heuristic detects the front most free space in the chamber, i.e. the lowest gap, in step (i). The first ship in the ordered list with a width smaller than or equal to the gap is selected for placement. It should be recalled that this list contains only ships that are currently under consideration for placement in the chamber. In step (ii), this ship will first be placed at the left-hand side of the gap (Figure 3.5 (a)). If this results in a feasible placement, the heuristic proceeds to step (i). When the left-hand side placement leads to a constraint violation (for example: Figure 3.5 (b), the current ship is not allowed to moor to the ship at the left-hand side of the gap), the heuristic proceeds to step (iii), where the ship is placed at the right-hand side of the gap. In case a feasible placement is obtained, the heuristic will proceed to step (i). If this alternative placement also leads to an infeasible solution, the heuristic selects the next ship in the ordered list that fits the width of the gap, and proceeds to step (ii). If at any time during the search none of the remaining ships fit the width of the gap, the gap is filled up to the level of the least-protruding gap-defining-ship and stored as wasted space

(Figure 3.5 (c) and (e)). The search procedure then proceeds to step (i) and it continues until all ships are placed. Afterwards, the occupied chamber length is checked. If this length is smaller than or equal to the length of the chamber, the obtained (feasible) solution is returned. Otherwise, a second attempt is made at solving the problem by ordering the ships by decreasing length. The constructed solution is returned if feasible. Otherwise, a final attempt is made by ordering the ships by decreasing surface. The alternative orderings are applied only when the previous ones have failed to construct a feasible solution. Hence, the computational cost of applying multiple orderings is very small compared to using only one ordering. This is illustrated with a small example: The first three ships can be placed in a feasible way using the decreasing width order. Hence neither the decreasing length nor the decreasing surface orderings will be used. When adding a fourth ship, the heuristic fails to find a feasible solution. It makes a second attempt by applying the decreasing length order, which yields a feasible solution. The algorithm continues by adding a fifth ship. The algorithm now fails when applying the decreasing width, decreasing length and decreasing surface orders, and returns the four-ship-solution. Section 3.7 shows that this multi-order approach does indeed generate better results than any of the orderings on their own, while the computational cost remains very small.

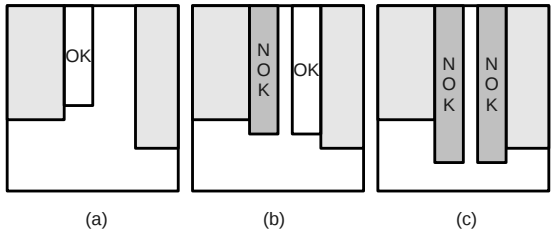

Figure 3.6: Example of the steps taken when placing a ship in the gap. OK indicates a feasible position for the ship, NOK an infeasible one.

Algorithm 2 Pseudo code of the multi-order best-fit heuristic for the lock scheduling problem using the leftmost placement policy.

```
Input: lockage L
Input: ship to add S
 1: if L empty then
 2:     L ← S at position (0,0)
 3:     return L
 4: else
 5:     while next ordering policy Ordering available do
 6:         newL ← new Lockage
 7:         ShipList ← all ships in L
 8:         ShipList ← S
 9:         order ships using Ordering
10:         while not all Ships added to newL do
11:             Find Lowest Gap
12:             while next Best-Fitting Ship available do
13:                 if Ship can be moored to the left gap defining ship then
14:                     newL ← Ship at leftmost position in gap (Subfigure a)
15:                     Raise chamber skyline to reflect addition of Ship
16:                     goto 10
17:                 else if Ship can be moored to the right gap defining ship then
18:                     newL ← Ship at rightmost position in gap (Subfigure b)
19:                     Raise chamber skyline to reflect addition of Ship
20:                     goto 10
21:                 end if
22:                 Current ship cannot be placed in the gap (Subfigure c)
23:             end while
24:             Raise Gap to Lowest Neighbour
25:         end while
26:         if Packing length ≤ Chamber length then
27:             return newL
28:         end if
29:     end while
30:     set newL infeasible (No feasible packing could be produced)
31:     return newL
32: end if
```

3.7 Experiments

We compare the performance of the three approaches on a set of instances generated based on real data. First, the two exact solution methods for the SPP are compared to display the tremendous reduction in computation time obtained by applying the FIFO based decomposition. Next, the performance of different settings for the multi-order best-fit is reviewed to identify the most interesting best-fit configuration with respect to solution quality, while showing the limited influence of additional orderings on the heuristic's computation time. Finally, the results of the multi-order best-fit heuristic are compared with the results obtained by the exact decomposition approach on two test sets to show each method's solution quality, computational cost and practical applicability.

3.7.1 Experimental setup

Both inland (single-traffic) and mixed-traffic data sets are used in the experiments. The first test set contains examples of ship arrivals for an inland setting, with a wide variety of properties. These properties are the number of ships in the problem, the size of the chamber, the sizes of the ships and the order in which the ships need to be placed. Both the chamber sizes and ship dimensions correspond to the actual dimensions of the locks and traffic on the Albertkanaal (Belgium). The ship dimensions have been randomly sampled from all traffic on the Albertkanaal over a 12 month period. In this test set, the ships are allowed to moor to any other ship. Thus, the set $MOOR_i$ will contain all ships that are not shorter than ship i. We assume that all safety distances are already part of the ship and chamber dimensions. Therefore, ship dependent safety distances are not used in the inland test set. The test set is available online [Verstichel, 2012]. Ten instances were generated for each instance size. The small instances (containing between 10 and 25 ships) were used for the experiments in Section 3.7.2 only. The properties of the inland instances are:

- Number of ships: 10, 20, 50, 100, 200, 500, 1000, and 10 to 25
- Chamber size: 16m x 136m, 24m x 200m
- Ship size: between 4.25m x 16.27m and 10.50m x 110m

The first mixed-traffic test set has been extracted from one month of actual lockage operations in the port of Antwerp. Ship size dependent safety distances must be taken into account, and two different ship types are considered. The presence of tugboats, which leave the chamber before the lockage operation starts, adds an extra dimension to the safety distances. No ships are allowed to moor to a 'SEA' ship, which can only moor directly to the quay, while 'BARGE' ships may moor to another barge. The chambers correspond to configurations of the Berendrecht (BE), Zandvliet (ZV), Boudewijn (BO), Van Cauwelaert (VC), Royers (RO) and Kallo (KL) locks. The test instances can be obtained upon e-mail request to the author. Some figures about the mixed-traffic test instances include

- Total number of ships: ~ 9000
- Number of ships per lockage: 1 to 18
- Chamber size: between 35m x 270m and 68m x 500m
- Ship size: up to 43m x 350m

The second mixed-traffic test set has been extracted from one month of actual lockage operations at the lock complex of Terneuzen (The Netherlands). Ship type dependent safety distances must be taken into account, and two different ship types are considered. No ships are allowed to moor to a 'SEA' ship, which can only moor directly to the quay, while 'BARGE' ships may moor to another barge. The chambers correspond to configurations of the Westsluis (W), Oostsluis (O) and Middensluis (M) locks. Some figures about these mixed-traffic test instances include

- Total number of ships: ~ 5400
- Number of ships per lockage: 1 to 11
- Chamber size: between 24m x 140m and 38m x 290m
- Ship size: up to 36m x 229m

All experiments were performed on a Dell Optiplex 790 with an Intel® Core™ i7-2600 (3.40GHz) and 8GB of memory running a 64-bit Linux Mint. The multi-order best-fit heuristic was implemented using Sun JDK 1.6, using the data structures of the three-way best-fit heuristic (Section A.2). The optimal solutions were obtained with Gurobi 5.1 under an academic license, with a time limit of 1 hour.

3.7.2 Exact approaches

We compare the multi-lockage SPP model from Section 3.3 with the decomposed single-lockage model from Section 3.5. Both models always yield an optimal solution to the ship placement problem. The exact position of each ship in the chambers, however, may differ between the two approaches. Both models were first evaluated on a set of inland instances with 10 to 13 ships, transferred by a single large chamber (24m x 200m). For larger instances, the original model frequently required more than 1 hour to attest optimality. Table 3.1 shows that the decomposed model exploiting the FIFO constraints is significantly faster than the full model. This indicates that generating a feasible solution to many single-lockage ship placement problems is easier than solving a single multi-lockage instance to optimality. Consider, for example, a 13 ship instance from Table 3.1 with a known upper bound of 5 required lockages. The experiments show that the corresponding 13 single-lockage ship placement problems are solved much faster than the single 5 lockage instance. In the second experiment, we compare the time required by the full model to discover an optimal solution, and the time require to prove its optimality. This is achieved by providing the

full model with a lower bound, generated from the optimal solutions of the decomposed model. Adding this lower bound to the model drastically reduced the computation time and enabled solving all instances with up to 21 ships in less than 1 hour. For instances with 22 and more ships, optimality was often not reached in less than one hour, and for sizes 26 and above the full model frequently failed to find an initial feasible solution in the same amount of time. Figure 3.7 shows a summary of this experiment, plotting the computation time of the full model with and without attesting optimality, and the decomposed approach, whereas Table 3.2 contains detailed information for each instance size. All other experiments are based on the decomposed exact approach only. Figure 3.8 shows the solution of one specific ship placement problem, where 10 ships are placed in one single chamber. This solution was found in 1 second and leaves less than 10% of the chamber's surface free.

Table 3.1: Comparison of the average and maximum calculation time in seconds for the full and decomposed models. Ten instances were solved for each problem size.

	Average time (s)		Maximum time (s)	
#Ships	Full	Decomp	Full	Decomp
10	8.55	0.44	30.01	1.29
11	25.30	0.31	121.26	0.73
12	61.80	0.52	393.33	2.19
13	167.32	1.53	988.74	11.83

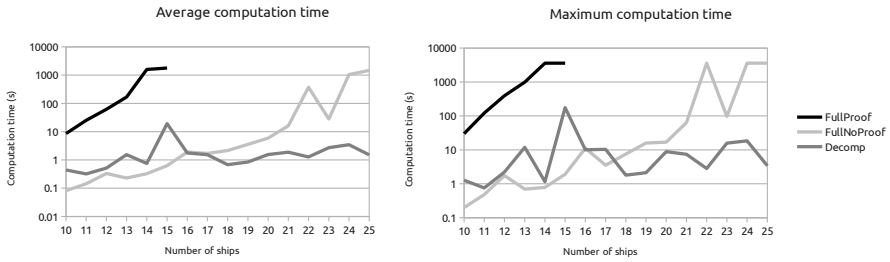

Figure 3.7: Comparison of the computation time required by three exact approaches. 'FullNoProof' shows the time required by the full model to find the optimal solution, while 'FullProof' also includes the time required to prove optimality. 'Decomp' shows the time required by the decomposition approach to find and prove the optimal solution.

Table 3.2: Comparison of the number of feasible ('Feas') and optimal ('Opt') solutions generated and the average calculation time required in seconds ('Time'). 'FullNoProof' shows the results for the full model when finding the optimal solution suffices, while 'FullProof' shows the results when optimality also has to be attested. 'Decomp' shows the results of the exact decomposition approach. Ten instances were solved for each problem size.

	FullProof		FullNoProof		Decomp	
#Ships	Feas(Opt)	Time	Feas(Opt)	Time	Feas(Opt)	Time
10	10(10)	8.6	10(10)	0.1	10(10)	0.4
11	10(10)	25.3	10(10)	0.1	10(10)	0.3
12	10(10)	61.8	10(10)	0.3	10(10)	0.5
13	10(10)	167.3	10(10)	0.2	10(10)	1.5
14	10(7)	1585.8	10(10)	0.3	10(10)	0.8
15	10(6)	1790.5	10(10)	0.6	10(10)	19.2
16			10(10)	1.9	10(10)	1.8
17			10(10)	1.7	10(10)	1.5
18			10(10)	2.1	10(10)	0.7
19			10(10)	3.6	10(10)	0.9
20			10(10)	5.9	10(10)	1.6
21			10(10)	16.1	10(10)	1.9
22			10(9)	373.0	10(10)	1.3
23			10(10)	28.1	10(10)	2.7
24			10(9)	1047.0	10(10)	3.5
25			10(7)	1465.7	10(10)	1.5
26			4(4)	2282.0	10(10)	0.9
27			2(2)	2901.7	10(10)	1.4
28			0(0)	3600.0	10(10)	3.5
29			0(0)	3600.0	10(10)	52.2
30			0(0)	3600.0	10(10)	6.7

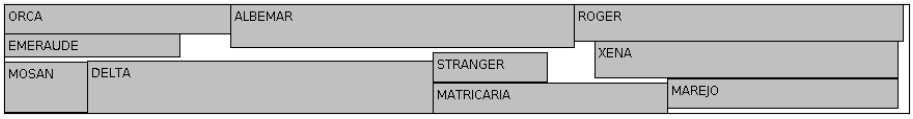

Figure 3.8: Example of a ship placement solution with 10 ships where less than 10% of the chamber's surface is free.

3.7.3 Multi-order best-fit heuristic

The multi-order best-fit heuristic was applied to the inland test instances and the results of a single-ordering best-fit are compared with the results obtained after applying three orderings (Section 3.6). The single-ordering heuristics are stripped versions of the multi-order best-fit heuristic, based on a single ship ordering. The results over all inland test instances for a single large chamber lock are shown in Table 3.3, which also contains the required number of lockages and the total calculation time needed for solving the entire test set. These results clearly indicate that the combination of multiple orderings outperforms the heuristics based on one single ordering. The calculation time needed by the multi-order best-fit heuristic stays well below the combined calculation times of the single-ordering best-fit heuristics. This is due to the conditional application of the alternative orderings, i.e. only when another ordering fails. The results for a single small chamber lock have been omitted because the heuristics generated a different number of lockages for a few instances only.

We conclude from these experiments that applying alternative orderings when necessary leads to an improved solution quality compared to applying one single ordering. Furthermore, this quality increase comes at a low computational cost. Finally, we point out that the multi-order best-fit heuristic is significantly better than the best of the single-ordering heuristics. In other words, conditionally applying the alternative ordering strongly outperforms selecting the best of the three single-ordering heuristics ($p - value < 10^{-4}$).

Table 3.3: Results of the multi-order best-fit heuristic with different orderings for a single large chamber lock. Results are averaged over 10 instances for each problem size. Best results are presented in bold. BF denotes best-fit, preceded by the applied ordering. '$\overline{\#}_l$' denotes the average number of lockages required over 10 instances of the given size.

	Width BF		Length BF		Surface BF		Multi-order BF	
#Ships	$\overline{\#}_l$	time (s)	#	time (s)	$\overline{\#}_l$	time (s)	$\overline{\#}_l$	time (s)
10	**2.1**	0.001	**2.1**	0.001	**2.1**	0.002	**2.1**	0.001
20	**3.9**	0.001	4	0.001	**3.9**	0.001	**3.9**	0.001
50	9.4	0.001	9.4	0.002	**9.3**	0.001	**9.3**	0.002
100	18.5	0.002	18.5	0.003	18.2	0.002	**18.1**	0.003
200	36.9	0.005	37	0.005	36.8	0.005	**36.3**	0.006
500	91.2	0.010	91.7	0.010	90.6	0.009	**89.6**	0.014
1000	179.3	0.020	180	0.020	178.8	0.021	**176.6**	0.029

3.7.4 Inland traffic

This series of experiments compares two approaches to the inland setting:

- Decomposed MILP approach (Section 3.5),
- Multi-order best-fit heuristic (Section 3.6).

We performed the experiments on all inland test instances and the results are presented in Table 3.4.

The multi-order best-fit heuristic reaches optimality with respect to the required number of lockages for 35 out of 70 instances, while requiring just a fraction of the time needed by the MILP approach. The computation times for the individual instances reveal that, on average, the best-fit heuristic consumes 0.6% of the time required by the MILP approach, with a minimum of 0.001% and a maximum of 22.5% on an instance containing 100 and 10 ships respectively. For the smaller instances (< 200 ships), the difference in number of lockages between the optimal solution and the best-fit heuristic was at most 1 on a total of 4. For the larger (≥ 200 ships) instances this difference was at most 7 lockages out of 173 or 4%, while the average optimality gap was 3.24%.

The number of lockages needed by the MILP approach and the multi-order best-fit heuristic were equal for all instances when using the small chamber. Due to these small differences, the results for this test setting were omitted.

Table 3.4: Results of the different ship placement algorithms, computed for a single large chamber lock. '$\overline{\#}_l$' denotes the average number of lockages required over 10 instances of the given size, while '$\#_{opt}$' denotes the number of optimal solutions found by the multi-order best-fit heuristic. '%Gap' denotes the average optimality gap. All results are averaged over 10 instances for each problem size.

	MILP		Multi-order best-fit			
#Ships	$\overline{\#}_l$	Time (s)	$\overline{\#}_l$	$\#_{opt}$	%Gap	Time (s)
10	2.1	0.449	2.1	10	0.00%	0.001
20	3.8	1.553	3.9	9	3.33%	0.001
50	9.2	6.950	9.3	9	1.11%	0.002
100	17.6	53.368	18.1	5	2.85%	0.003
200	35.0	24.328	36.3	2	3.68%	0.006
500	87.0	49.300	89.6	0	3.00%	0.014
1000	171.4	114.335	176.6	0	3.04%	0.029

3.7.5 Mixed traffic

The first mixed-traffic experiment compares the single-order and multi-order best-fit heuristics with the MILP approach on a number of queue instances. These instances were selected from the first port data set where a queue of ships was waiting to be transferred by the lock, and where the optimal solution contained at least one ship more than the arrival-order best-fit solution. This increasing arrival ordering may be preferred in port environments, as it places the first arriving ships at the front most positions in the chamber, making them the first to leave the chamber after the lockage operation is completed. Such single ordering heuristic may, however, lead to significantly worse results than a combination of orderings. Four orderings are applied for the multi-order best-fit heuristic: increasing arrival time, and decreasing width, length and surface. The decomposed MILP approach was applied with a time limit of 1 hour per iteration. The objective is to place as many ships as possible in the chamber under a first-come-first-served restriction.

The results from Table 3.5 show that ignoring the arrival order of the ships when selecting their position leads to a significantly higher occupation of the locks. The exact approach is frequently able to find better solutions than the heuristics, but requires significantly more computation time. Setting a lower time limit for the exact approach may bring the computation times closer together, but also increases the risk of missing the optimal solution. Figure 3.9 shows an example of how the usage of the multi-order best-fit heuristic outperforms the stand-alone increasing arrival ordering.

Table 3.5: Comparison of the average performance of the increasing arrival time order best-fit heuristic (Arrival BF), multi-order best-fit heuristic (Multi-order BF) and decomposed MILP approach (Exact) when ships are queued at the lock. The solution properties are average number of ships per lockage ('$\overline{\#}_s$'), average computation time ('Time') and the number of improved instances ('Impr') with the total number of instances for each lock between parentheses. *The time limit of 1 hour was reached for one instance.

	Arrival BF		Multi-order BF			Exact		
Lock	$\overline{\#}_s$	Time	$\overline{\#}_s$	Impr	Time	$\overline{\#}_s$	Impr	Time
BO	10	0.5 ms	11.8	5(6)	1.0 ms	12.7*	4(6)	1070.4* s
VC	8.6	2.6 ms	9.5	6(8)	4.6 ms	10.1	4(8)	165.4 s
RO	6.1	0.3 ms	6.7	3(9)	0.5 ms	7.3	6(9)	10.5 s
KL	7.6	0.7 ms	8.5	5(11)	1.6 ms	9.5	9(11)	3.8 s

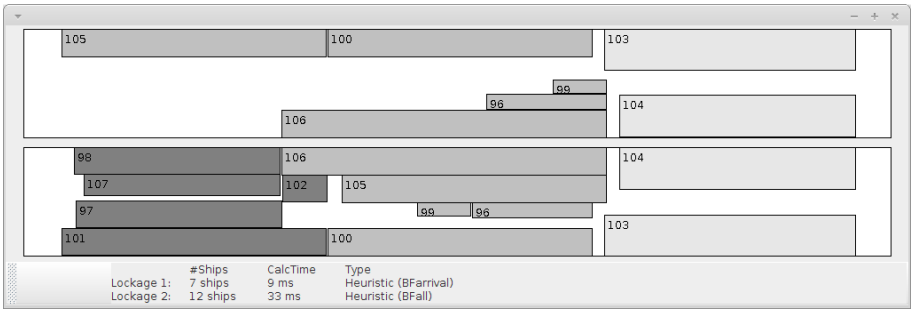

Figure 3.9: Example of an instance for which the multi-order best-fit (lower solution) strongly outperforms a single increasing arrival ordering best-fit (upper solution).

The second mixed-traffic experiment compares the algorithmic results with the manual solutions generated by the lock masters. The available data contained sets of ships that were processed together in manually obtained lockages from both mixed-traffic data sets. If a solution approach cannot place all ships from such a set in a single lockage, we consider it an algorithmic failure. Table 3.6 shows the number of manual lockages that could be reconstructed by each solution method, and the average computation time needed. The results show that the multi-order best-fit heuristic is able to reconstruct 99.36% of the solutions, while the exact approach reconstructs all lockages. The exact solutions do however come at a computational cost: while the heuristic solves each instance in less than 20 milliseconds, the exact approach requires up to 14 seconds for a single instance. The single-order best-fit heuristic performs only slightly worse than the other approaches, managing to reconstruct 98.53% of the instances in 0.1 milliseconds on average.

Table 3.6: Comparison of the increasing arrival time order best-fit heuristic (Arrival BF), multi-order best-fit heuristic (Multi-order BF) and decomposed MILP approach (Exact) when reproducing real-life lockages. '$\#_l$' denotes the number of lockages that could be reproduced, with the total number of lockages between parentheses, and Time shows the average computation time.

	Arrival BF		Multi-order BF		Exact	
Lock	$\#_l$	Time	$\#_l$	Time	$\#_l$	Time
BE	357(362)	0.22 ms	360(362)	0.29 ms	362(362)	11.07 ms
ZV	390(390)	0.08 ms	390(390)	0.08 ms	390(390)	5.89 ms
BO	397(405)	0.07 ms	401(405)	0.07 ms	405(405)	55.68 ms
VC	343(354)	0.08 ms	344(354)	0.07 ms	354(354)	7.54 ms
KL	545(546)	0.06 ms	546(546)	0.06 ms	546(546)	3.24 ms
RO	608(614)	0.05 ms	611(614)	0.04 ms	614(614)	2.93 ms
W	648(651)	0.18 ms	650(651)	0.18 ms	651(651)	2.67 ms
O	933(964)	0.14 ms	954(964)	0.15 ms	964(964)	3.42 ms
M	549(555)	0.10 ms	554(555)	0.10 ms	555(555)	1.41 ms
Total	98.53%	0.10 ms	99.36%	0.11 ms	100.00%	14.39 ms

3.8 Conclusion

In this chapter the ship placement problem, which is an important part of the lock scheduling problem, was presented. It has been identified as a variant of the 2D rectangular single bin size bin packing problem (2D rectangular SBSBPP) with additional constraints, and was proven to be different from 2D rectangular SBSBPP. A fast exact decomposition algorithm to the ship placement problem was developed by exploiting the precedence constraints of the ship placement problem. This algorithm enables solving problem instances with up to 1000 ships to optimality in less than 500 seconds by the general purpose solver Gurobi 5.1. These relatively long computation times, which sometimes differ strongly among similar instances, are a drawback for real-life applications. The ship placement problem may need to be solved many times while searching for a good solution for the entire lock scheduling problem and therefore short and stable calculation times are required. The multi-order best-fit heuristic addresses this issue. It constructs several possible SPP solutions based on different ship orderings and placement policies. When some ordering fails to produce a good solution, the multi-order best-fit heuristic applies an

alternative ordering in search for a feasible solution. A significant performance increase over each individual ordering is thus gained, while very little additional calculation time is required. The multi-order best-fit heuristic was compared to the exact decomposition approach on a large test set, showing that the heuristic is significantly faster than the exact approach, generating solutions for large instances with up to 1000 ships in less than 0.1 seconds with an average optimality gap of 3.24%. Both approaches were applied to real-life mixed-traffic instances from a port, on which they performed excellently. The multi-order best-fit heuristic was able to reconstruct 99.36% of the lockages performed by the lock masters, while never requiring more than 6 milliseconds of calculation time. The exact approach reconstructed all lockages, but did require up to 89 seconds to do so. When ships were queueing at the lock, the exact approach reached the time limit of 1 hour on one instance and calculated for more than 10 minutes in three cases, while several other instances were solved in less than 0.1 seconds. These unpredictable and sometimes long computation times are an obvious downside of this exact approach, especially for real-life applicability (e.g. as part of a decision support tool). The combination of high solution quality and low calculation times of the multi-order best-fit heuristic, on the other hand, make it a promising approach to the ship placement problem. These properties should be especially beneficial in a decomposition scheme for the lock scheduling problem where the ship placement problem constitutes the sub problem (Chapter 5). An additional characteristic of the multi-order best-fit heuristic is that it tends to group ships of similar size. This is perceived as a benefit because of the similarity with results obtained by human experts who currently solve the ship placement problem in real-life situations. Consequently, the solutions generated by the multi-order best-fit heuristic are easier for the human experts to analyse, thus increasing acceptance of the presented solution.

Chapter 4

Decision support for lock masters: a case study

Currently lock masters schedule lock operations with little or no support from optimization software. Their tasks are, however, far from trivial. Positioning ships in a chamber, for example, is already NP-hard (Section 3.1). Although the number of ships that can be transferred together in a single lockage operation is limited, the ship placement instances lock masters are faced with on a daily basis can be quite challenging.

The potential benefits of adding decision support software to the lock master's tool suite are identified and evaluated by means of live-tests at mixed-traffic locks. The live-tests focus solely on the ship placement problem because it constitutes a difficult task for lock masters, especially during peak traffic hours when increasing numbers of ships want to be transferred together while the time available for constructing a good solution decreases. Last-minute changes due to ships cancelling or delaying their arrival and vessels showing up at the lock without advance notice can disrupt a meticulously planned lockage. Live-tests have a double advantage: they enable a thorough test of a tool's real-life applicability, and the interaction with the end-users creates a perfect setting for identifying possible improvements and resolving some previously unknown issues. Furthermore, these tests constitute an additional evaluation of the real-life applicability of the solution approaches for the ship placement problem from Chapter 3. Although the experiments from Chapter 3 were performed on a large set of historical mixed-traffic data, they ignored the stochastic aspects that are typical for real-life operations.

For the purpose of these live-tests a decision support tool for lock masters was developed: MOGLi[1]. In this tool incorporates the solution approaches from Chapter 3 in an intuitive graphical user interface, enabling lock masters to easily generate and compare several ship placement solutions based on a list of arriving ships. Furthermore, all relevant data such as ship arrivals, lock configurations and parameters such as safety distances and computation times can easily be manipulated. Ships that can be transferred in the same lockage are automatically selected and positioned in a feasible and easy to understand way. The tool is highly configurable, allowing lock masters to quickly evaluate the effect of an alternative chamber for transferring some of the arriving ships, changing the lock configuration or altering the order in which the ships are processed. MOGLi's user interface and configuration options are discussed in Appendix B.

The chapter starts with an informal and high-level description of some lock master's tasks at a (parallel chamber) lock. The second part of the chapter reports on live tests of MOGLi at the Boudewijn-Van Cauwelaert lock complex in the Port of Antwerp and at the lock complex of Terneuzen (The Netherlands).

4.1 Decision support for lock masters

A lock master is responsible for all operations of a designated lock. He decides which chamber a ship will be transferred in, at which position the ship will be moored in the chamber, and at what time the lockage operation will take place. The lock master also handles all transfer requests of barges, taking into account predetermined lockage times for some high priority ships (e.g. seagoing vessels) that are pre-assigned to the lock chambers by the port-wide control centre. During periods of low density traffic, the lock master is confronted with a relatively easy problem: positioning a few ships inside a chamber is easy and oftentimes at least one chamber will be available for transferring ships almost as soon as they arrive. In addition the lock master has the time to construct a good solution as he receives a limited number of transfer requests from arriving ships. Under normal operations, both the time at which a lockage will take place and the chamber in which a lockage will be scheduled are often a result of the previous lock operations, leaving the positioning of the arriving ships as the most difficult part. The lock master's tasks are especially challenging under peak traffic. Not only is a good ship placement solution significantly more difficult to produce when the number of ships increases, the time available for constructing such a solution decreases because the lock master's line of thought

[1]Multilevel Optimization and Generic models for Lock operations

is frequently interrupted by requests from arriving ships and actions required for handling the current lockage operations.

A decision support system automatically suggesting feasible ship placement solutions would be a most welcome addition to the lock master's tools, especially during peak and normal traffic periods.

4.2 Live-tests

In light of a practical validation of this doctoral research, the opportunity arose to perform live-tests of the ship placement algorithms at the Boudewijn-Van Cauwelaert lock complex in the Port of Antwerp (Belgium) and at the lock complex in Terneuzen (The Netherlands). During these tests MOGLi ran in parallel with the lock master, proposing lockages for the arriving traffic based on the arrival-order best-fit heuristic, multi-order best-fit heuristic and exact solution method. The solutions generated by these three algorithms were displayed together, enabling a quick and easy comparison of each solution by the lock master. We will first give an overall evaluation of the three solution methods, followed by an analysis of the tool's behaviour during low density, normal and peak traffic.

4.2.1 Solution approaches

During the tests it became clear that the exact solution method produces solutions that are hard to grasp. Ships sometimes appeared to be placed at curious, though feasible, positions in the chamber. Even though this way of placing the ships may enable placement of additional ships, the resulting solution would never be accepted by the lock master for two reasons: the lock master would have a hard time certifying that the solution is feasible, not in the least because he has limited time to analyse the proposed solution, and the captains/barge operators would likely refuse to position themselves at unconventional locations in the chamber. Furthermore, the unpredictable computation times of this approach are not favourable in a time constrained environment. The solutions generated by the multi-order best-fit heuristic, on the other hand, were deemed sound as the reason why a ship is positioned in a certain way is easy to understand, enabling a quick and easy attestation of the solution's quality and feasibility. The short and stable computation times of this heuristic was also considered beneficiary.

One downside of the current solution approaches is their lack of support for convoys. Convoys consisting of an odd number of units are often L-shaped instead of rectangular shaped, leaving a space for placing a ship as shown in Figure 4.1. Currently the real shape of convoys is not exploited by the ship placement algorithms because the information required to incorporate convoy configurations in the solution approaches is not available. In the current tool suite, convoys are are also rectangular shaped, and a ship positioned in the convoy's gap will thus be overlapping with the convoy in the registration. It would therefore be interesting to add a convoy's shape or configuration to the current lock master's software, allowing the incorporation of convoys to our solution approaches.

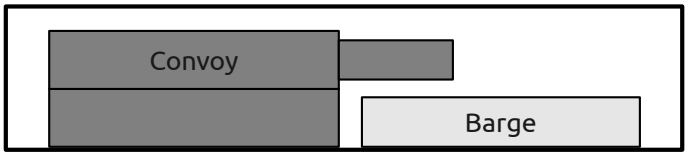

Figure 4.1: Example of the real shape of a convoy with an odd number of units, and the additional space that is freed when convoy shapes can be exploited.

4.2.2 Low density traffic

During periods of low density traffic the decision support software did not yield improvements. All proposed solutions (both by the lock master and MOGLi) were trivial, with the few ships transferred in each lockage occupying only a small portion of the chamber. These periods did allow the lock masters to become acquainted with MOGLi, test several of its features and provide valuable input for improvements to the software's user interface. One suggested improvement was the addition of a snap function with real-time indication of the ship's destination position when manually moving a ship inside a chamber.

4.2.3 Normal traffic

The periods of normal traffic enabled the decision support software to display its potential value. The computed solutions showed great similarity with or were even identical to the ones constructed by the lock master. The main difference between the manually constructed solutions and the ones generated by the multi-order best-fit heuristic was the computation time: where the lock master needed up to several minutes to construct a solution, the software tool produced its results in a few milliseconds. Figures 4.2 and 4.3 show instances where the

generated solution was identical to the one constructed by the lock master. The successful one-on-one comparison of both solutions strongly increased the lock master's trust in the decision support software, and before long the lock master started validating generated solutions instead of manually constructing them himself and afterwards comparing to the generated solutions.

Given the ease with which lockages are generated under normal traffic, the planning horizon of the lock could increase significantly. Since ships registering at the lock can effortlessly be added to a future lockage, all arriving ships could receive a rough estimate of their time of transfer upon enrolment based on the current registrations and the remaining free space in the lockage. If accurate enough, barge operators could change their sailing speed based on this estimated transfer time by either speeding up to catch an early lockage (and thus reduce their waiting time) or slowing down to arrive just in time for their lockage (and thus reducing their fuel cost and pollution).

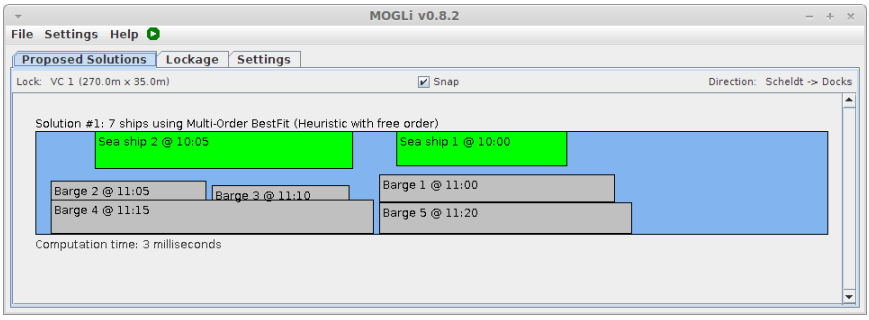

Figure 4.2: Screenshot of a solution generated by MOGLi, identical to the one constructed by the lock master.

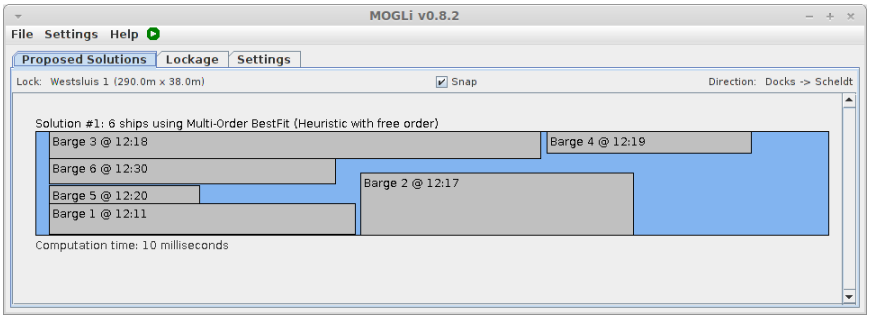

Figure 4.3: Screenshot of a solution generated by MOGLi, identical to the one constructed by the lock master.

4.2.4 Peak traffic

During peak traffic, the decision support software applying the multi-order best-fit heuristic was able to show its full potential. MOGLi successfully supported the lock master by generating high quality solutions in a rapidly changing environment. The impact of the decision support software under such circumstances is illustrated in the following paragraph.

Peak traffic occurred in the Port of Antwerp when thirteen barges requested to be transferred to the docks at approximately the same time. With the large Boudewijn chamber reserved for an incoming seagoing vessel and the smaller Van Cauwelaert planned in the opposite direction, the circumstances were less than ideal: Transferring as many ships as possible in the Boudewijn chamber, together with the incoming seagoing vessel, would induce a considerable waiting time of 30 to 45 minutes per barge. Changing the schedule of the Van Cauwelaert chamber would significantly reduce waiting time, but was only worthwhile if most of the barges could be transferred together in this smaller chamber. In both scenarios any barge that could not be transferred would have to wait at the lock for at least an additional half hour. At this point, the best approach would be to compare solutions for both scenarios. Manually constructing these solutions, however, is a time consuming process due to the large number of ships involved, and the Van Cauwelaert chamber had to be turned soon if the second scenario was to be executed. MOGLi, on the other hand, was able to produce solutions in a matter of milliseconds. For the Boudewijn-scenario, the decision support software suggested a solution transferring the seagoing vessel and all enrolled barges in a single lockage operation (Figure 4.4). Next, a solution for the Van Cauwelaert scenario was generated containing all but the latest registered ship *Barge 13* (Figure 4.5). After confirming the solution's feasibility, the lock master decided to transfer the barges in the Van Cauwelaert lock, thereby reducing the waiting time of all but the last arriving barge to a minimum. While copying the ship's positions to the lock registration software, however, the lock master got a message cancelling *Barge 12*'s arrival. Given that *Barge 13* is significantly larger than *Barge 12*, both ships could not simply be interchanged. Fortunately, removing the cancelled barge from the ship list allowed the decision support software to generate a, completely different, solution transferring *Barge 13* along with the eleven other registered barges in a matter of milliseconds. The execution of this lockage, which is added in Figure 4.6, resulted in a minimal waiting time for all registered barges despite the traffic peak and last-minute changes that occurred.

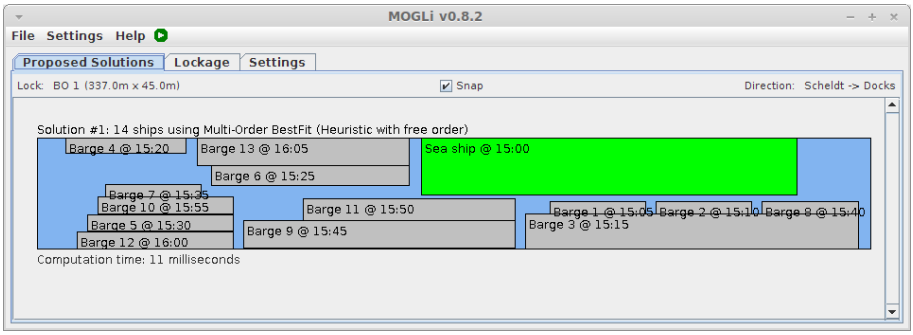

Figure 4.4: Solution generated by MOGLi for the Boudewijn scenario.

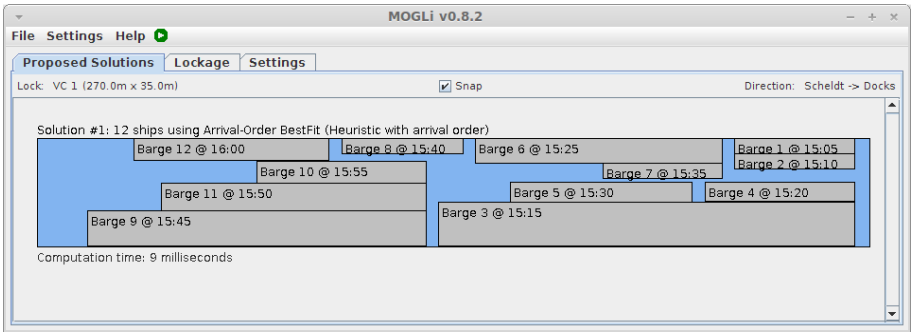

Figure 4.5: Solution generated by MOGLi for the Van Cauwelaert scenario.

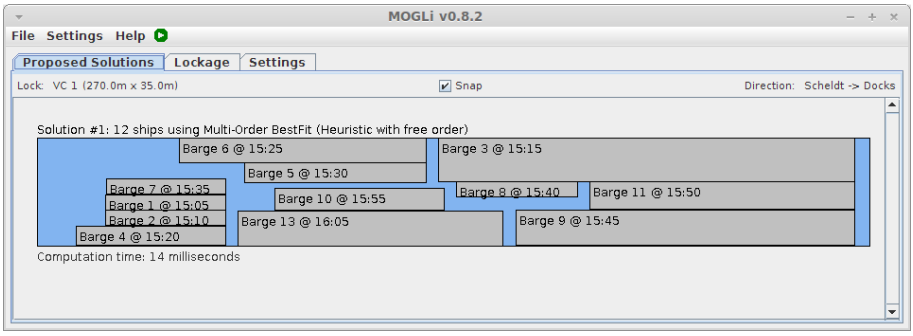

Figure 4.6: Solution generated by MOGLi for the Van Cauwelaert scenario after the last-minute cancellation of Barge 12.

4.3 Conclusion

Solving the ship placement problem is one of the daily task of lock masters. Altering an existing solution to incorporate last-minute changes and constructing solutions during periods of peak traffic can be a challenging task, especially given the tight time constraints. We investigated whether lock masters could benefit from the support of optimization software and if the solution approaches from Chapter 3 would also perform well in real-life, stochastic environments. Live-tests performed at two mixed-traffic locks showed that decision support software generating high quality ship placement solutions would indeed be a valuable addition to the lock master's current tools. It is of utmost importance that the applied algorithm generates solutions that are easy to grasp. The lock master must be able to quickly understand why each ship is placed at its position because he has limited time to analyse a suggested solution and deduce its feasibility. Failing to do so will result in the rejection of the proposed solution, significantly reducing the benefit of the decision support software and the lock master's trust in the proposed solutions. Short and predictable calculation times are also imperative given the time constrained nature of the problem. Furthermore, a faster solution generation leaves more time for the lock master for feasibility analysis and alterations to the configuration. While the software's assistance is especially valuable during peak traffic hours and in case of last-minute changes to ship arrivals, the time saved under normal traffic conditions could enable the lock master to focus on other important tasks and increase the planning horizon, improving the overall quality of service for the arriving ships.

It would be very interesting to include the scheduling and chamber assignment parts of LSP into future versions the decision support software for further live-testing. This could provide invaluable information about overall solution quality of current and future algorithms for the lock scheduling problem in real and stochastic environments, and their impact on lock efficiency, especially at multi chamber locks.

Chapter 5

A Combinatorial Benders' decomposition for the lock scheduling problem

The lock scheduling problem can be decomposed into a master and a sub problem. The master problem (MP) assigns the ships to lock chambers, after which it attempts to schedule the resulting lockage operations. The sub problem (SP) considers positioning the ships inside the corresponding lockages. Whenever the sub problem identifies an infeasible lockage, i.e. a set of ships that cannot be transferred simultaneously due to the chamber's capacity or safety constraints, combinatorial inequalities (cuts) are generated and added to the master problem. The master problem and sub problem are solved iteratively, until a provable optimal (and feasible) schedule is obtained.

This chapter mainly focuses on the decomposition approach and its application to the LSP, thereby omitting detailed discussions on the exact sub problems. First, Section 5.1 provides a literature review on an established exact decomposition approach. Section 5.2 presents a Benders' decomposition approach for LSP, thereby defining the master and sub problem in more detail, as well as their interaction. Particular attention goes to the generation of feasibility cuts for the master problem, as they largely determine the efficiency of the algorithm. Experiments are conducted on a large number of real-world instances based on data collected at several locks in Belgium. The results are presented in Section 5.3. Finally, Section 5.4 offers the conclusions.

5.1 Literature review

Benders' decomposition is a mathematical approach that exploits the fact that fixing a number of difficult variables in a mathematical model may simplify the problem considerably. The decomposition approach divides a problem into a master problem (MP) and a sub problem (SP) which are solved iteratively. First, the MP is solved for a subset of the variables. Next, the sub problem is solved for the remaining variables, while temporarily fixing the variable values of the MP. Finally, based on the outcome of the SP, one or more cuts are generated and added to the MP, thereby effectively preventing the MP from revisiting similar areas of the search space.

In traditional (classical) Benders' decomposition [Benders, 1962], the SP is a linear programming problem; cuts are derived from its dual solution. In more recent works, e.g. Geoffrion [1972] and Hooker and Ottoson [2003], the Benders' decomposition approach has been generalized to a broader class of problems, no longer requiring the sub problem to be linear. In Hooker and Ottoson [2003], the concept of Logic Based Benders' decomposition is introduced. In contrast to Benders [1962], cuts are not necessarily obtained from the dual formulation of a linear sub problem, but through the so-called inference dual. Whenever the sub problem is a feasibility problem, the inference dual is a condition which, when satisfied, implies that the master problem is infeasible [Rasmussen and Trick, 2007]. This condition can then be used to obtain Benders' cuts to cut off infeasible solutions. A particular case of Logic Based Benders' decomposition, frequently referred to as Combinatorial Benders' decomposition, is discussed in Codato and Fischetti [2006] where it is applied to mixed-integer programming (MIP) problems involving large numbers of logical implications (Big_M constraints). Whenever a particular assignment of variable values in the MP renders the SP infeasible, a Combinatorial Benders' cut is generated and added to the master problem. This cut, stating that at least one of the variables in the master problem must change its value, distils a logical implication from the original model and adds it to the master problem. Note that this approach is ineffective for continues variables. Stronger Combinatorial cuts may be obtained by identifying small subsets of variables responsible for the infeasibility of the sub problem, and expressing cuts in terms of these variables. The smallest of these subsets are referred to as Minimum Infeasible Subsets (MIS). The latter approach will be applied in this work, as will be elaborated on in Section 5.2.

A number of successful applications of Combinatorial Benders' decompositions to related packing, scheduling and assignment problems have been reported in literature. Bai and Rubin [2009] investigate a problem involving the allocation of tollbooths to roads, thereby minimizing the number of tollbooths required to cover the entire road network. Similar to our problem, their problem suffers

from a large number of conditional (Big_M) constraints. By decomposing the problem, many of these conditional constraints can be omitted. A master problem assigns the tollbooths to roads. Subsequently, the sub problem verifies whether the proposed assignment is feasible. Whenever an infeasible solution is encountered, cover cuts are generated, stating that at least one tollbooth must be placed on a specific subset of roads.
Côté et al. [2013] use Combinatorial Benders' decomposition to solve the Strip Packing Problem (SPP). First a relaxed version of SPP is solved, thereby treating SPP as a Parallel Processor Scheduling Problem. Next, the sub problem attempts to reconstruct a feasible solution to SPP based upon the relaxed solution. Whenever this is not possible, a combinatorial cut is added to the master problem, requiring that at least one rectangle must change its position. Strong cuts are obtained by identifying small subsets of rectangles responsible for the infeasibility of the sub problem.

A Parallel Machine Scheduling Problem (PMSP) with machine and sequence dependent setup times is solved in Tran and Beck [2012] through Logic Based Benders' decomposition. In this approach, the master problem assigns jobs to machines, while the sub problem minimizes the makespan for each individual machine by determining the job sequence. This is efficiently realized by a dedicated TSP solver. Note that, due to the fact that the machines are independent, these sequencing sub problems may be solved in parallel. To strengthen the master problem, a relaxed version of each sub problem is added. Based on a comparative study, Tran and Beck [2012] claim that their Benders' decomposition for the PMSP is 6 orders of magnitude faster than a traditional Branch-and-Bound approach.

5.2 A Combinatorial Benders' decomposition

In contrast to Chapter 2 where we attempted to solve the LSP via a single, large, mixed integer linear programming problem, we now propose to solve LSP by using a decomposition approach. The decomposition results in a master problem and a sub problem, each of which have to be solved iteratively. The advantage of this decomposition is that part of the complexity of the problem is shifted to a separate sub problem, thereby obtaining two simpler problems. In addition, efficient dedicated algorithms can be employed to solve these problems, whereas there may not exist an algorithm capable of tackling the entire problem at once. Finally, a number of logical implications which are modelled in Chapter 2 through Big_M constraints are no longer required, as these implications will be enforced through the addition of cuts to the master problem. The presented method is very similar to Codato and Fischetti [2006]. The main differences are

that we work with an integer programming sub problem instead of a linear one, and that we apply a constructive algorithm for determining *minimal infeasible subsets* (MIS), where Codato and Fischetti [2006] determined MIS through an LP.

The decomposition method provides the master problem with a list of ships that need to traverse the lock, the sailing direction of the ships, and their arrival times at the lock. The master problem first partitions the ships in an arbitrary number of non-overlapping subsets. Each set represents a group of ships that will be transferred in a single lockage operation. Obviously, each subset contains only ships that traverse the lock in the same direction. Next, for the generated subsets, the master problem proposes a schedule, thereby determining the exact starting times of the lockage operations. Subsequently, the sub problem verifies for each subset whether the ships in that set can be transferred simultaneously, i.e. whether they fit together inside the lock chamber. The latter corresponds to the ship placement problem from Chapter 3. LSP is solved whenever an optimal MP schedule is determined in which each subset satisfies the packing constraints of the sub problem. Whenever the sub problem identifies an infeasible combination of ships, a feasibility cut is generated and added to the master problem, thereby preventing the master problem from assigning these ships to the same lockage operation. Figure 5.1 shows where the master problem (grey-*italic*) and the sub problem (white-**bold**) are located in the representation of the mathematical model for the lock scheduling problem.

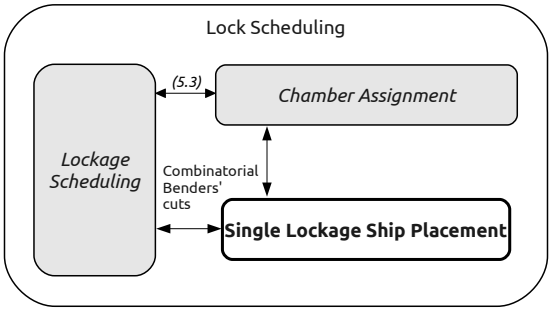

Figure 5.1: Position of the MP (grey-*italic*) and SP (white-**bold**) in the representation of the mathematical model for the lock scheduling problem.

The following two subsections discuss the master problem and sub problem in detail. An overview of the entire algorithm is given in Algorithm 3.

5.2.1 Master problem

The following MILP defines the master problem. To keep the model concise, some problem specific constraints were omitted, e.g. constraints managing tidal windows, ship dependent pre and post-processing times, ship draught, etc. Similarly, some redundant constraints to speed up the model are not included in the problem description below. The complete model was presented in Chapter 2.

$$min \quad \lambda_1 \sum_{k} \boldsymbol{z_k} + \lambda_2 \sum_{i \in N} wct_i \boldsymbol{c_i} + \lambda_3 \boldsymbol{T_{max}} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{k \in M^j} \boldsymbol{f_{ik}} = 1 \qquad \forall i \in N^j, j = 1, 2 \tag{5.2}$$

$$\boldsymbol{f_{ik}} \leq \boldsymbol{z_k} \qquad \forall i \in N, k \in M \tag{5.3}$$

$$\boldsymbol{z_{k+1}} \leq \boldsymbol{z_k} \qquad \forall k \in M_t, t \in T \tag{5.4}$$

$$\boldsymbol{c_i} \geq C_{max}(\boldsymbol{f_{ik}} - 1) + \boldsymbol{C_k} \qquad \forall i \in N, k \in M \tag{5.5}$$

$$\boldsymbol{P_k} \geq p_t \boldsymbol{z_k} + \sum_{i \in N} p_i \boldsymbol{f_{ik}} \qquad \forall k \in M_t, t \in T \tag{5.6}$$

$$\sum_{u \in U_t} \boldsymbol{proc_{ku}} = \boldsymbol{z_k} \qquad \forall k \in M_t, t \in T \tag{5.7}$$

$$\boldsymbol{proc_{ku}} + \sum_{v \in U_t, v \neq u} \boldsymbol{proc_{lv}} + \boldsymbol{seq_{kl}} \leq 2 \qquad \forall k, l \in M_t,\ l > k,\ \forall u \in U_t,\ t \in T \tag{5.8}$$

$$\boldsymbol{C_l} - \boldsymbol{C_k} + 2C_{max}(3 - \boldsymbol{seq_{kl}} - \boldsymbol{proc_{ku}} - \boldsymbol{proc_{lu}}) \geq \boldsymbol{P_l} + s_{kl} \qquad \forall k, l \in M_t, l > k, \forall u \in U_t, t \in T \tag{5.9}$$

$$\boldsymbol{C_k} - \boldsymbol{C_l} + 2C_{max}(2 + \boldsymbol{seq_{kl}} - \boldsymbol{proc_{ku}} - \boldsymbol{proc_{lu}}) \geq \boldsymbol{P_k} + s_{lk} \qquad \forall k, l \in M_t, l > k, \forall u \in U_t, t \in T \tag{5.10}$$

$$\boldsymbol{C_k} \geq r_i \boldsymbol{f_{ik}} + \boldsymbol{P_k} \qquad \forall i \in N, k \in M \tag{5.11}$$

$$\boldsymbol{T_{max}} \geq \boldsymbol{c_i} - r_i \qquad \forall i \in N \tag{5.12}$$

The master problem uses four sets of variables: binary variables $z_k, k \in M$, denoting whether lockage $k \in M$ is used, binary variables $f_{ik}, i \in N, k \in M$ denoting whether ship $i \in N$ is assigned to lockage $k \in M$, integer variables $P_k, k \in M$ denoting the processing time of lockage $k \in M$ and finally integer variables $C_k, k \in M$ denoting the completion time of lockage $k \in M$.
The objective, Equation (5.1), minimizes (a) the number of lockages, (b) the time when a ship leaves the lock and (c) the maximum waiting time of a ship at the lock, where $\lambda_1, \lambda_2, \lambda_3$ are independent weight factors. Constraints (5.2)-(5.4) assign ships to lockage operations. Constraints (5.2) ensure that each ship is assigned to a lockage operation. Obviously, downstream ships cannot be assigned to upstream lockages and vice versa. Constraints (5.3) are linking constraints; a lockage $k \in M$ is executed, i.e. $z_k = 1$, if at least one ship is assigned to it. Note that the lockage operations are ordered (Constraint (5.4)): a lockage z_{k+1} cannot be executed if z_k is not used, $k \in M_t, t \in T$. The remaining constraints take care of the scheduling part of the problem. A ship cannot leave the lock before its lockage operation is completed (Constraints (5.5)). The duration of a single lockage operation depends on a fixed value plus an additional amount per ship (Constraints (5.6)). Each lockage operation must be mapped to a physical lock chamber (Constraints (5.7)). When two lockage operations are scheduled on the same physical chamber, one must precede the other (Constraints (5.8)). Constraints (5.9), (5.10) perform the actual lockage scheduling per chamber (explained in Chapter 2). A lockage cannot commence before all ships have arrived at the lock (Constraints (5.11)). Finally, Constraint (5.12) records the maximum waiting time of a ship at the lock.

5.2.2 Sub problem

Once the master problem has assigned the ships to a number of lockages, the feasibility of each lockage needs to be verified. For each lockage, i.e for all $k \in M_t : z_k = 1$, $t \in T$, a small sub problem is solved to test whether the given configuration of ships fits inside a chamber of type t. Whenever a configuration is considered infeasible, a combinatorial Benders' cut will be generated and added to the master problem. The latter will be elaborated in Sections 5.2.3 and 5.2.4.
Let $\overline{N_k} = \{i \in N : f_{ik} = 1\}$ be the set of ships assigned to lockage $k \in M$. For a given lockage $k \in M$, we obtain the following single-lockage ship placement problem:

$$\boldsymbol{x_i} + w_i \leq W_t \qquad \forall i \in \overline{N_k} \qquad (5.13)$$

$$\boldsymbol{y_i} + l_i \leq L_t \qquad \forall i \in \overline{N_k} \qquad (5.14)$$

$$\boldsymbol{left_{ij}} + \boldsymbol{left_{ji}} + \boldsymbol{b_{ij}} + \boldsymbol{b_{ji}} \geq 1 \qquad \forall i < j,\ i,j \in \overline{N_k} \qquad (5.15)$$

$$\boldsymbol{x_i} + w_i \leq \boldsymbol{x_j} + W_t(1 - \boldsymbol{left_{ij}}) \qquad \forall i \neq j,\ i,j \in \overline{N_k} \qquad (5.16)$$

$$\boldsymbol{y_i} + l_i \leq \boldsymbol{y_j} + L_t(1 - \boldsymbol{b_{ij}}) \qquad \forall i \neq j,\ i,j \in \overline{N_k} \qquad (5.17)$$

safety constraints $\qquad \forall i \neq j,\ i,j \in \overline{N_k} \qquad (5.18)$

mooring constraints $\qquad \forall i \neq j,\ i,j \in \overline{N_k} \qquad (5.19)$

In the above feasibility model, variables $\boldsymbol{x_i}, \boldsymbol{y_i}$ model the x and y coordinates of a ship $i \in \overline{N_k}$ inside the chamber. Constraints (5.13) and (5.14) ensure that the x and y coordinates of a ship $i \in \overline{N_k}$ are located within the chamber's dimensions. The remaining constraints ensure that the ships do not overlap. Several problem specific constraints have been omitted here to avoid repetition. The full sub problem is presented in Section 3.5.

Algorithm 3 Combinatorial Benders' Decomposition of LSP

Input: Set of ships N, arrival times and ship properties, lock parameters

add initial cut(s) to MP
repeat $\leftarrow$ true
while repeat **do**
  Solve MP
  get solution $(\boldsymbol{z_k}, \boldsymbol{f_{ik}}, \boldsymbol{C_k})$, $\forall i \in N, k \in M$
  repeat $\leftarrow$ false
  for $k \in M : z_k = 1$ **do**
    Solve SP for $\overline{N_k} = \{i \in N : \boldsymbol{z_{ik}} = 1\}$
    if SP is infeasible **then**
      repeat $\leftarrow$ true
      add feasibility cut(s) to MP
    else
      get solution $(\boldsymbol{x_i}, \boldsymbol{y_i})$, $\forall i \in \overline{N_k}$
    end if
  end for
end while
return Optimal schedule $(\boldsymbol{f_{ik}}, \boldsymbol{C_k}, \boldsymbol{x_i}, \boldsymbol{y_i})$, $\forall i \in N, k \in M$

5.2.3 Combinatorial Benders' cuts

When an infeasible sub problem is encountered, one or more combinatorial Benders' cuts are generated and added to the master problem, effectively preventing the master problem from assigning specific ships to the same lockage.

The general form of cuts considered is:

$$\sum_{i \in S \subseteq N} \boldsymbol{f_{ik}} \leq |S| - 1 \qquad \forall k \in K' \subseteq M \tag{5.20}$$

Stated informally, this cut prevents the ships in $S \subseteq N$ from being assigned to the same lockage k.

A straightforward 'no-good' cut arises from an infeasible sub problem, i.e. an infeasible lockage of type $t \in T$ with $\overline{N_k}$ ships, by setting $S = \overline{N_k}$ and $K' = M_t$. These no-good cuts can be very weak, especially if $|S|$ is large. Stronger cuts may be obtained when smaller infeasible subsets of ships are considered. The strongest cuts are based on minimum infeasible subsets. In this context, a MIS is a subset of ships that cannot be transferred in a single lockage of type $t \in T$; removing any of the ships from the set would however result in a feasible lockage. Computing all MIS for a given set of ships $N' \subseteq N$ is unfortunately notoriously difficult; in fact, it would require solving the sub problem from Section 5.2.2 for every possible subset of N'. Section 5.2.4 discusses several approaches to compute strong cuts, requiring far less computational effort.

Combinatorial Benders' cuts can be generated at different times in the solution process: *Initial cuts* are added to strengthen the master problem before the MP solution process is started. Applying initial cuts reduces the number of infeasible MP lockages generated. *Feasibility cuts* are generated on-the-fly every time the master problem generates a solution. These cuts are applied to cut away infeasible parts of the search space and guide the MP towards a feasible lock scheduling solution.

5.2.4 Cut separation

The different cut separation methods are clarified using the example from Figure 5.2, where the MP proposes a solution in which ships 1 through 7 are transferred in a single lockage. The feasible lockages for this example (under a first-come-first-served policy) are displayed on the right side of this figure.

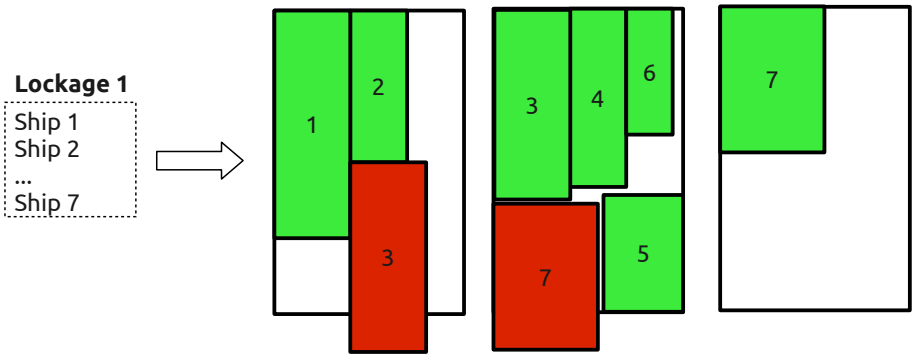

Figure 5.2: An example where the MP proposes a solution where ships 1 through 7 are transferred in a single lockage, and the feasible first-come-first-served based lockages.

No-good cuts can easily be computed by solving one single-lockage ship placement problem for each MP lockage. For the example from Figure 5.2, the following weak cut is generated:

$$\sum_{i=1}^{7} f_{ik} \leq 6, \quad \forall k \tag{5.21}$$

Minimal infeasible subsets can be found by applying the following constructive procedure. For a given set of ships, all subsets of size n are calculated, where n is initially set to 2. The sub problem (Section 5.2.2) is solved for each of these subsets, and a feasibility cut is generated when necessary. Next, all subsets of size $n+1$ are generated and compared against the infeasible subsets generated in the previous iterations. Every subset of size $n+1$ which is a superset of a MIS generated in a previous iteration is discarded. For the remaining sets N' of size $n+1$, the sub problem is solved and cuts are generated where applicable. The constructive procedure terminates whenever no new cuts can be identified (i.e. all generated subsets of size n are infeasible). Note that the larger the number of ships that can be transferred in a single lockage, the more computationally expensive this procedure becomes. Cuts produced by this procedure will be

referred to as 'subset cuts'. For the example in Figure 5.2, thirteen subset cuts are generated.

An alternative means to generate minimal infeasible subsets of ships utilizes a strict ordering on the ships. Let $N' = \{1, 2, \ldots, n'\}$ be an ordered set of ships, based on their arrival time. Start by setting $S = \{1\}$. Iteratively add ships from the head of N' to S until S becomes an infeasible subset of ships. This is an 'order cut'. When generating *feasibility* cuts, all ships in S are removed from N', except the last ship added to S, and the procedure is repeated until N' is exhausted. When generating *initial* cuts, only the first ship in S is removed from N' before the procedure is repeated. Note that these cuts are particularly effective under a FCFS lockage policy. When considering the example from Figure 5.2, two feasibility order cuts can be generated:

$$\sum_{i=1}^{3} f_{ik} \leq 2, \quad \forall k \tag{5.22}$$

$$\sum_{i=3}^{7} f_{ik} \leq 4, \quad \forall k \tag{5.23}$$

An efficient approach for identifying *small* infeasible subsets of ships is based on surface calculations: any set of ships having a combined surface that exceeds the total surface of the lock chamber is infeasible. Whenever surface calculations are used to identify infeasible subsets, it will be denoted as follows: 'subsurf' (subset based) and 'surf' (order based). As simple surface calculations provide a non-tight upper bound on the number of ships that can be placed, they can be applied as initial cuts only. Indeed, applying them as feasibility cuts does not guarantee that the MP will converge towards a feasible solution, as surface based cuts may be unable to cut away some infeasible parts of the search space.

5.3 Experiments

The performance of the decomposition method and the 'monolithic' MILP approach from Chapter 2 are compared. The test set consists of inland traffic data and different lock settings based on the locks on the Albertkanaal. All instances are available online [Verstichel, 2012] together with the lock parameters. The test set contains the random-arrival (R) instances, and their properties are added to Table 5.1. The independent weight factors are: $\lambda_1 = 0.1$, $\lambda_2 = 1.0$ and $\lambda_3 = 1.0$.

All experiments were performed on a Dell Optiplex 790 with an Intel® Core™ i7-2600 (3.40GHz) and 8GB of memory running a 64-bit Linux Mint. The

monolithic approach, MP and SP were solved using Gurobi 5.1 under an academic license, with a total time limit of 12 hours.

Table 5.1: Properties of the test instances.

Inter arrival time distribution	Uniform (R)
Mean inter arrival time (minutes)	1,2,3,4,5,10,15,30
Number of ships	10, 20, 30, 40, 50, 60, 70, 80, 90
Upstream/Downstream fraction	50/50, 30/70
Small Chamber	16m x 136m
Large Chamber	24m x 200m

5.3.1 First-come-first-served single chamber lock

The first series of experiments is performed on a single chamber lock with a first-come-first served policy for the ships; both a single small chamber (SSC) and a single large chamber (SLC) are considered. The experiments assess the performance of the feasibility cuts and the effects of adding some initial cuts to the MP. The results are depicted in Figure 5.3 and 5.4. The x-axis of each figure displays the different instances which are ordered, from left to right, based on (1) increasing number of ships (2) increasing inter arrival time and (3) traffic ratio (first 70/30, then 50/50). The numbers underneath the axis are formatted as I_S, with I denoting the inter arrival time, and S the number of ships. Computation times in seconds are shown on the y-axis of each figure (logarithmic scale). Note that these computation times include the generation of both the feasibility and initial cuts.

The most basic version of the Benders procedure relies on no-good cuts only and is referred to as 'no-good' in the graphs. A stronger version is obtained by replacing the no-good cuts by order cuts (Figure 5.3 (a)). Especially for the larger instances, a significant decrease in computation time is observed. Another approach is to generate a number of initial cuts and add them to the initial MP. Figure 5.3 (b) reveals a drastic reduction of computation times with such initial cuts. Here, applying no-good feasibility cuts without initial cuts (no init) is compared with combining no-good cuts with initial surface cuts (surf init) or order cuts (order init). When considering the SLC setting (Figures 5.4 (a) and (b)), the difference between the cuts becomes much smaller. Contrary to the SSC results, the average inter arrival time also seems to influence the computation time: for the small instances (< 30 ships) the computation time decreases when the average inter arrival time increases, while the opposite trend shows for the large instances (≥ 40 ships).

In Figure 5.3 (c), the results obtained using the Benders' approach in combination with initial order cuts are plotted against the monolithic approach for a large number of instances. Clearly, the former method outperforms the latter. Especially for several of the larger instances, where computation times are reduced by 95%. The other approaches shown in Figures 5.3 (a,b) could not outperform the monolithic model and were therefore omitted from the graph.

The differences between the monolithic model and the Benders' decomposition approach become more profound in the SLC setting (Figure 5.4 (c)). The largest difference in computation time is observed for the instance with 20 ships, $\sigma = 2$ and symmetric traffic: the monolithic model timed out after 12 hours, whereas the Benders procedure applying initial order cuts attested optimality in only 1.3 seconds. The maximum computation time for the Benders approach for instances with up to 40 ships is 12 minutes, while the monolithic model fails to solve 16 out of 64 instances within 12 hours. For the instances with 50 and 60 ships, the monolithic model could only solve a single instance to optimality and failed to produce a feasible solution for 7 out of 32 instances. The decomposition approach on the other hand finds feasible solutions for all instances, and attests optimality in 24 cases. Finally, for the instances consisting of 70 to 90 ships, the decomposition method solves 21 out of the 48 instances to optimality while feasible solutions were found for the remaining instances.

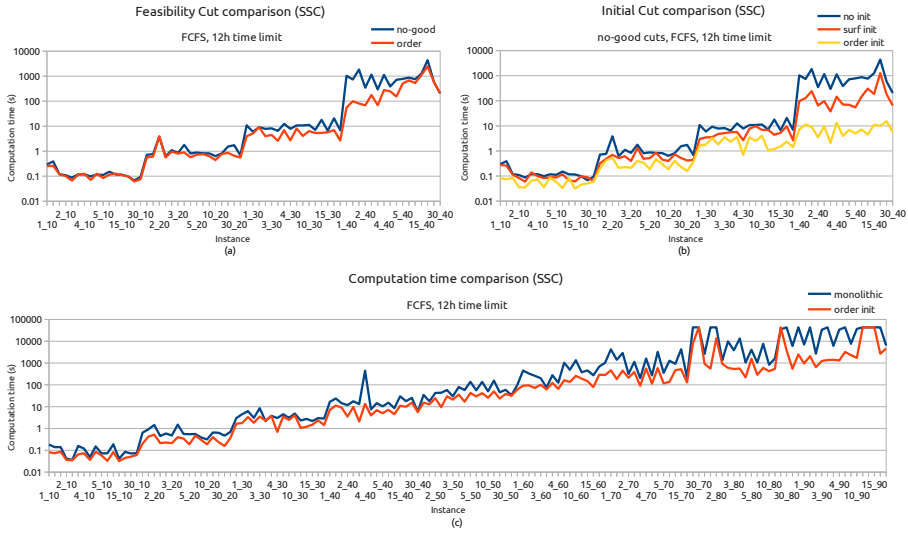

Figure 5.3: Comparison of the computation time of the different cut generation methods for a single small chamber lock, under a FCFS policy.

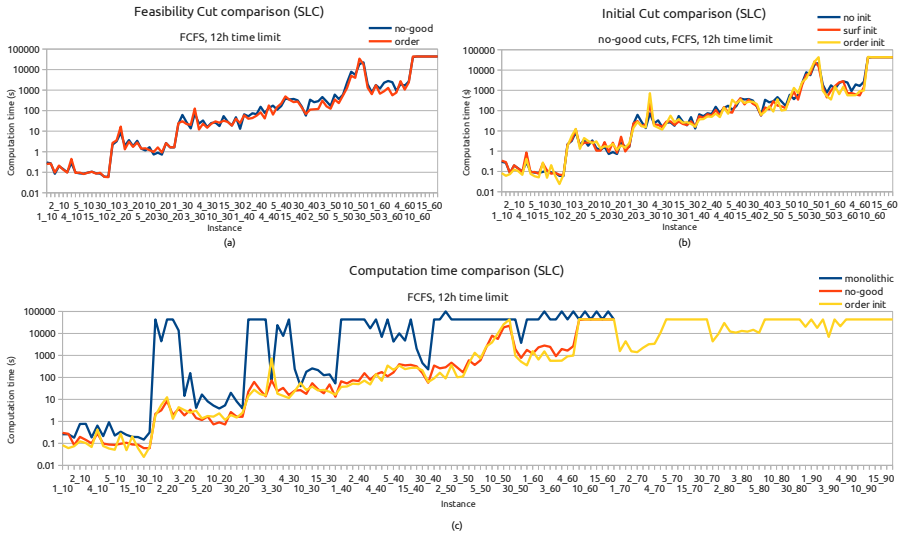

Figure 5.4: Comparison of the computation time of the different cut generation methods for a single large chamber lock, under a FCFS policy.

5.3.2 No first-come-first-served single chamber lock

The second series of experiments is conducted with the same traffic and lock data, but without the first-come-first-served policy. The results for instances with 10 and 30 or more ships were omitted: all 10 ship instances were solved in less than 2 seconds and only a few of the large instances were solved in less than 12 hours.

Dropping the FCFS policy has several implications for the decomposition method. In the MP, a number of constraints are dropped as they no longer apply in the absence of the FCFS policy. Consequently, the MP becomes substantially harder to solve. Furthermore, the absence of an explicit ordering of the ships permits a significantly larger number of ship to lockage assignments in the MP, rendering the no-good and order cuts ineffective. The resulting performance decrease is reflected by the number of MP-SP iterations: the instances from Section 5.3.1 are solved within a few iterations, whereas the same instances in the absence of the FCFS rule require up to 3200 iterations when applying no-good or order cuts. In the subsequent experiment, these cuts have been replaced by the computationally more expensive subset cuts. Figures 5.5 (a) and (b) compare the performance of various methods. For the small

chamber lock, using initial subset cuts appears to be the best approach, while for the larger chamber, a combination of initial order cuts and feasibility subset cuts works best. In either case, the decomposition approaches outperform the monolithic model by a large extent.

From the above results, it is apparent that the absence of the FCFS rule has a significant impact on the computation times. We therefore adopt the FCFS policy for the remaining experiments.

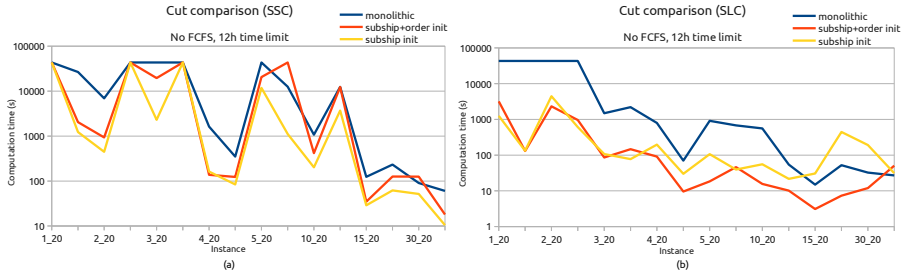

Figure 5.5: Comparison of the computation time of different approaches for single chamber locks without FCFS.

5.3.3 First-come-first-served parallel chamber lock

For the identical parallel chamber instances, the results for instances with 10 and ≥ 30 ships were omitted. For a lock with two small parallel chambers, the difference between the initial cuts is limited (Figure 5.6 (a)). Applying either no initial cuts or subset initial cuts results in the best overall performance. Figure 5.6 (b) shows that the feasibility subset and initial subset cut generation methods are slower than the monolithic approach when inter arrival times are large. The decomposition method is almost always faster for the other instances, with a total computation time of 6.5, 2 and 1.5 hours for the monolithic, feasibility subset and initial subset approaches respectively. For a large parallel chamber lock, the initial surface cuts and initial order cuts show the best overall performance, closely followed by the initial subsurface cuts (Figure 5.7 (a)). All other initial cuts are significantly slower than the aforementioned initial cuts. When comparing with the monolithic approach (Figure 5.7 (b)), the results show that the feasibility and initial subset cut generation methods are much faster when the ship inter arrival time is short. For instances with medium inter arrival time (~ 10 minutes) no clear winner can be found, while the monolithic approach is the best choice when facing large inter arrival times. Furthermore,

both cut generation methods attested optimality on one instance more than the monolithic approach.

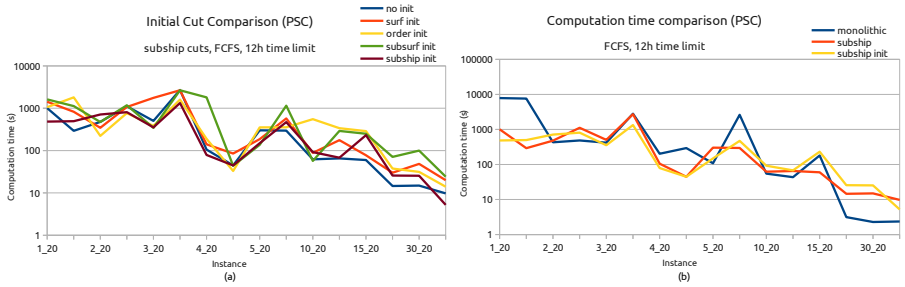

Figure 5.6: Comparison of the computation time of different cut generation methods for a small parallel chamber lock under a FCFS policy.

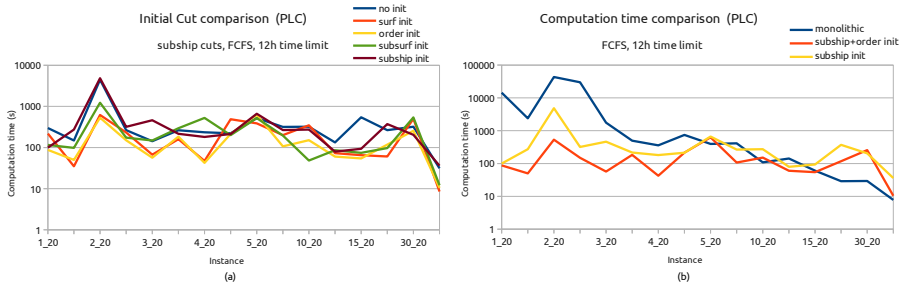

Figure 5.7: Comparison of the computation time of different cut generation methods for a large parallel chamber lock under a FCFS policy.

5.3.4 First-come-first-served multi chamber type lock

The results for the multi chamber type lock are summarized in Figure 5.8. Here only the ≥ 30 ship instances were omitted. Similar to the SSC results it appears that, aside from the number of ships, the ship inter arrival time has the largest influence on the required computation time. Applying initial subset cuts appears to be the best way of tackling LSP for this multi chamber type setting. It is the fastest approach on all but a few instances and is able to attest optimality for 31 out of 32 instances, while the monolithic approach fails to do so for 10 instances.

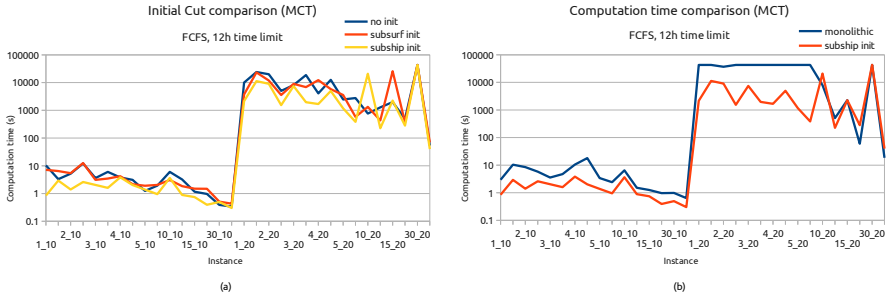

Figure 5.8: Comparison of the computation time of different cut separation methods for the multi-chamber type lock under a FCFS policy.

5.3.5 Heuristic sub problem solution method

The last experiment considers the effects of applying the multi-order best-fit heuristic to the SP while reducing the total calculation time to ten minutes. The results of this heuristic approach are summarized in Table 5.2, with the results of the monolithic approach under a time limit of 10 minutes added between parentheses. When applying initial order cuts on the single small chamber lock, all instances with less than 70 ships were solved to optimality in less than 5 minutes, whereas the monolithic approach failed to produce a feasible solution in two cases. For the larger instances the heuristic approach matched the exact results on 26 out of 48 instances, while in 4 cases no initial solution could be produced within the time limit. The monolithic approach was not able to produce more than 16 feasible and 8 exact results in the same amount of time. For the single large chamber lock, the average optimality gap on the instances with less than 70 ships was 0.21% for the heuristic approach. There are two classes of instances for which optimality was not reached. A first class was solved in less than 10 minutes and therefore optimality was unattainable on these 23 instances due to the application of the multi-order best-fit heuristic which was unable to produce the optimal ship placement solutions. For the second class of instances the time limit was reached, leaving two possible origins for the remaining optimality gap: either the multi-order best-fit heuristic was unable to produce optimal ship placement solutions, or the master problem had not converged after 10 minutes and a further improvement of the objective would be obtained when increasing the time limit. When applying the monolithic model, 31 instances could not be solved and the exact results were matched on 51% of the instances. The heuristic approach obtained better results than the exact decomposition approach for two instances with 70 ships. For the first

instance, the maximum waiting time was reduced by 31 minutes, while the total waiting time and the number of lockages remained the same. A reduction of the total waiting time by 33 minutes was witnessed for the second instance, at the expense of one additional lockage and an increase of the maximum waiting time by 10 minutes. Both approaches reached the time limit of 12 hours (exact) and 10 minutes (heuristic) on these instances. Applying the exact decomposition method to both instances with an increased time limit enabled the reproduction and improvement of the heuristic solutions.

All PSC instances were solved to optimality by the heuristic decomposition method when applying initial subset cuts, and the monolithic approach matched the heuristic's results on all but a few instances. For the PLC setting the multi-order best-fit heuristic prevented the decomposition method from constructing the optimal solution for two instances, resulting in a small optimality gap of 0.04% and 0.12%. Similar results are obtained for the multi chamber type lock, where optimality could not be obtained for four instances, leading to a maximum gap of 0.45%.

These results show that applying a high-performance heuristic for solving the sub problem has very little impact on solution quality. The large differences in solution quality between the exact and heuristic approaches for the very large instances can be fully attributed to the MP which is unable to find a good overall solution in 10 minutes. Furthermore, the heuristic decomposition approach produced significantly better results than the monolithic model when computation time is limited.

Table 5.2: Summary of the heuristic experiments. Results for the monolithic approach with a time limit of 10 minutes are added between parentheses as a reference. '# ships' denotes the range in instance size for the row and 'Total' the total number of instances in this range. The number of feasible solutions found by the heuristic decomposition approach is added under 'Feasible'. 'Exact' shows the number of instances for which the exact solution was matched by the heuristic approach while 'Gap' shows the average gap between the exact and heuristic solutions. *The heuristic decomposition outperformed the exact solution approach for two instances.

Lock	# ships	Total	Feasible	Exact	Gap
SSC	10-60	96	96 (94)	96 (93)	0.00%
	70-90	48	44 (16)	26 (8)	17.60%
SLC	10-60	96	96 (65)	60 (49)	0.21%
	70-90	48	48 (0)	3* (0)	76.10%
PSC	10-20	32	32 (32)	32 (31)	0.0%
PLC	10-20	32	32 (32)	30 (31)	0.01%
MCT	10-20	32	32 (32)	28 (27)	0.03%

5.4 Conclusion

An exact decomposition method for the lock scheduling problem has been introduced. By solving the scheduling and assignment parts of LSP in the master problem and dealing with the ship placement aspect of LSP in the sub problem, an efficient solution method for the lock scheduling was constructed. Not only are most Big_M constraints removed from the model, thereby enabling a much tighter linear relaxation of the master problem, the resulting sub problem is also one for which some very effective (exact) solution methods exist. Through the application of efficient cut separation methods for several types of combinatorial Bender's cuts in a branch-and-bound scheme, numerous new optimal results are obtained for instances with up to 90 ships. The decomposition approach is most effective when several ships can be transferred in a single lockage operation, i.e. when the solution to the packing part of LSP is non-trivial. Reducing the computation time from twelve hours to ten minutes and applying a heuristic to the sub problem results in high quality solutions for all but the largest instances, proving the potential of the proposed solution method in a heuristic approach. These results also pinpoint the MP procedure as the main bottleneck in the presented decomposition approach. Future work will therefore be aimed at

improving the MP procedure. For example by improving the convergence rate of the MP by adding a number of valid inequalities, or by applying a heuristic method for the MP.

Chapter 6

Conclusion

The lock scheduling problem is a complex combinatorial optimization problem: ships need to be assigned to lock chambers, their exact position inside the chambers has to be defined and the resulting lockages require scheduling. Given that locks are a key component of tide independent ports and numerous inland waterways, optimization of lock operations could positively influence the efficiency and economical attractiveness of water-bound transportation.

The lock scheduling problem for locks with a single or multiple parallel chambers was studied in this thesis. The primary goal of this thesis was reached through accurate modelling of the lock scheduling problem and its sub problems, which enabled the formulation of real-life applicable exact and heuristic solution methods. The development of a Combinatorial Benders' decomposition method capable of solving the lock scheduling problem by means of exact and heuristic solution methods fulfilled this thesis' secondary objective. Furthermore, we showed that this complex combinatorial problem can be solved efficiently in real-life situations, and that decision support software based on the developed solution approaches could be a valuable addition to a lock master's tool suite.

6.1 Contributions

The generalized lock scheduling problem for single-traffic and mixed-traffic locks was introduced and modelled as a mixed integer linear programming problem incorporating a wide range of properties and constraints from practice. As a result, real-life feasible and, under the given objective, optimal solutions could

be generated for small instances.

The ship placement sub problem of the LSP was studied extensively and a decomposition method was presented enabling the generation of optimal solutions to instances of any size in reasonable albeit somewhat unpredictable time. A constructive heuristic for the ship placement problem generating near-optimal results in a matter of milliseconds was developed and tested extensively. The heuristic's short and stable computation time and its high solution quality render it ideal for application in decomposition methods where the ship placement problem constitutes the sub problem or in decision support software.

The ship placement solution approaches were incorporated in a decision support tool for lock masters that was evaluated during live-tests at a major port and on important mixed-traffic locks. The assistance from decision support software was proven to be especially effective under peak traffic and in case of last-minute changes to the ship arrivals.

An accessible application of Combinatorial Benders' decomposition to the lock scheduling problem was presented, along with several efficient cut generation methods. By solving the assignment and scheduling parts of the LSP in the master problem and checking the resulting lockage's feasibility through the application of a ship placement solution method, even very large problem instances could be solved to optimality in less than twelve hours. Applying the multi-order heuristic to the ship placement sub problem enabled solving nearly all large instances to optimality in less than ten minutes, proving once more the heuristic's excellent solution quality. For the very large instances, the slow convergence of the master problem prevented the algorithm from reaching satisfying results.

6.2 Future research directions

The lock scheduling problem offers many opportunities for future research. While some research directions concern new and faster solution methods, the problem can also be extended to incorporate other lock configurations and some strongly connected problems that were not addressed in this thesis.

Currently, the practical applicability of the presented decomposition approach for the lock scheduling problem is limited by the slow convergence of the exact solution approach for the master problem. The development of a faster, possibly heuristic, solution method for this scheduling problem would facilitate solving large and very large instances in acceptable computation time.

Several combinations of exact and heuristic MP-SP solution approaches can be applied in the presented Combinatorial Benders' decomposition scheme. While the application of exact solution approaches to both the master and the sub problem was tested extensively and some experiments were performed with heuristic solution methods for the packing sub problem, it would be most interesting to compare these approaches while applying a heuristic solution method to the master problem.

Where this thesis focussed on a single lock with multiple chambers, the situation may arise where a ship can choose between several locks at different geographical locations serving the same body of water. One example is the port of Antwerp where three locks transfer ships from the river Scheldt to the port and back: the Berendrecht-Zandvliet lock, the Boudewijn-Vancauwelaert lock and the Royers lock. Taking into account travel times from a ship's current position to each lock and from these locks to the ship's destination would enable co-scheduling the locks and possibly further increase overall efficiency.

While the solution methods in this thesis enable optimizing the operations at an individual lock, inland waterways often consist of a series of locks. Scheduling these sequential locks together while taking into account travel times between reaches (the parts of the inland waterway between two locks) and ships entering or leaving the traffic flow between two locks would be an interesting approach and could enable the introduction of a 'green wave' for barges, allowing them to transfer a series of locks without any delays.

Tugboat planning would also be a most interesting addition to the lock scheduling problem. The effects of an optimal lock scheduling could be reduced when too few tugboats are available to move all transferred ships to their destination. Coupling lock scheduling and tugboat planning would be especially interesting when tugboats can be interchanged between locks, or when a ship's travel time depends upon the number of tugboats assisting it.

The applicability of the developed Combinatorial Benders' decomposition method is not limited to the lock scheduling problem. The method has potential to improve existing approaches to other problems combining scheduling/planning and packing (the patient admission scheduling problem with room assignment constraints), assignment and packing (the shelf space optimization problem), ...

Appendix A

The three-way best-fit heuristic

The contribution of this appendix, which is based on [Verstichel et al., 2013c], is an improved best-fit heuristic, called the three-way best-fit heuristic. This heuristic is transformed for application to the ship placement problem in Chapter 3 of this thesis. Due to the differences between the orthogonal strip packing problem and the ship placement problem (e.g. no rotation, limited number of ships per lockage, ...) important parts of the paper could not be discussed in Chapter 3.

The three-way best-fit heuristic applies a combination of the ideas from the original best-fit heuristic [Burke et al., 2004] in conjunction with an efficient input sequence disruption. In Section A.1, the three-way best-fit heuristic is introduced and the ambiguities concerning the placement of rectangles are clarified. Following this, the time complexity of the heuristic is improved by implementing the data structures from Imahori and Yagiura [2010] in Section A.2. Section A.3 discusses the results of the heuristics, in respect to both solution quality and computation time. Finally, in Section A.4 the conclusions are presented.

A.1 A three-way best-fit heuristic

Throughout the section which follows hereunder, a small example will be used to clarify the different item orderings and selection procedures. The example

contains three rectangles with the following widths w_i and heights h_i: 40x16, 25x24 and 16x13.

A.1.1 Original best-fit heuristic

This section presents the best-fit heuristic developed by Burke et al. [2004]. It introduces its essential components which will be extended and refined throughout the appendix. The pseudo code of the heuristic is presented in Algorithm 4, while the important concepts of the heuristic are depicted in Figure A.1. Figure A.1 (b) shows the skyline, while Figure A.1 (d) shows the location of the gaps and the lowest gap.

Given an initial list of n rectangles, the heuristic first generates a list of size $2n$, containing all rectangles in both their default and rotated configuration. The $2n$ rectangles are then ordered by decreasing horizontal dimension. For the small example, this results in the following item order: 40x16, 25x24, 24x25, 16x40, 16x13 and 13x16. Upon the conclusion of this step, the solution construction begins by initialising the sheet skyline. This is essentially an integer list with a length equal to the sheet width. Each element of this list contains the present total height of the packing (initially 0) at its x coordinate. Once the skyline is initialised, the lowest gap is located, i.e. the lowest sequence of x coordinates with an identical height. The heuristic then determines the best-fitting rectangle for this gap by iterating over the ordered rectangles. This best-fitting rectangle is the first rectangle in the list fitting the width of the gap. When the rectangle does not fit the gap exactly, it will be placed at either the left or the right side of the gap, based on the decision of the placement policy used. Once the best-fitting rectangle is placed, the skyline is adjusted to reflect this addition, and the rectangle's two configurations are removed from the list. In the case that no best-fitting rectangle can be found for the current gap (i.e. its width is smaller than the smallest rectangle dimension), the skyline at the gap is lifted so that it levels with the lowest of the rectangles neighbouring the gap. The process described above is repeated until all the rectangles are placed.

After the construction phase, the post processing part of the heuristic attempts to further improve the solution's quality by removing towers. This is achieved by first checking whether the topmost rectangle is placed in its rotated configuration, in which case the rectangle is rotated by 90 degrees and placed at the lowest possible position on the sheet. If this renders an improvement, the process is repeated for the new topmost rectangle. When this procedure does not constitute an improvement, or when the topmost rectangle has already been placed in its default configuration, the post processing step terminates.

Algorithm 4 Pseudo code of the best-fit heuristic as introduced in Burke et al. [2004].

```
Input: Stock sheet dimensions
Input: List of n rectangles to pack
Rotate each rectangle so that width ≥ length
Sort rectangle list by decreasing rectangle width
for Each placement policy do
   Initialize skyline
   while not all Rectangles packed do
      Find Lowest Gap
      if Best-Fitting rectangle available then
         Place rectangle using Placement Policy
         Update skyline
      else
         Raise Gap to Lowest Neighbour
      end if
   end while
   while TowerReduction not finished do
      Find topmost rectangle
      if Rectangle width ≥ length then
         Optimization finished
      else
         Remove topmost rectangle
         Rotate rectangle
         Update skyline
         while Rectangle doesn't fit Lowest Gap do
            Raise Gap to Lowest Neighbour
         end while
         Place rectangle using Placement Policy
         Update skyline
         if Packing not improved then
            Optimization finished
         end if
      end if
   end while
end for
```

A.1.2 Original placement policies

The best-fit heuristic uses three different placement policies:

- Leftmost: place the fitting rectangle at the left-hand side of the gap
- Tallest: place the fitting rectangle adjacent to the tallest gap-defining rectangle, with preference for the sheet side
- Shortest: place the fitting rectangle next to the shortest gap-defining rectangle

Burke et al. [2004] assume that the shortest neighbour policy is the inverse of the tallest neighbour policy. Careful examination of their results, however, reveals that whenever confronted with two neighbours that end at the same level, both placement policies add the rectangle at the left-hand side of the gap. Only when the lowest gap is defined by the sheet sides will the shortest neighbour policy place the rectangle at the right-hand side, which is indeed the inverse of the position the tallest neighbour policy would adopt.

Before starting the introduction of the new placement policies, we first present a more precise definition of the tallest and shortest neighbour policies used in Burke et al. [2004], in order to avoid any possible ambiguity. This definition is based on careful analysis of the reported results, policy descriptions and figures from the original paper.

The tallest neighbour policy places the new rectangle next to the gap-defining rectangle that ends at the highest level, disregarding the actual height of this rectangle. From the point of view of the new rectangle, this is its tallest neighbour. The sheet side is always the tallest neighbour, as the sheet is considered to be infinitely long. In case of a tie, the tallest neighbour policy always places the rectangle at the left-hand side of the gap. This placement policy is visualised in Figure A.1, where the new rectangle is always shown in grey. Figure A.1 (a) shows a tie, as the gap is flanked by both sheet sides. Therefore, the rectangle is placed at the left-hand side. The rectangle added in Figure A.1 (b) fits exactly, so no placement decision has to be made. In A.1 (c), the sheet side is the tallest neighbour, so the rectangle is placed at the left-hand side of the gap. In A.1 (d) the rectangle fits exactly, while in A.1 (e) the left sheet side is the tallest neighbour. Figure A.1 (f) represents a tie as both gap-defining rectangles end at the same height, thus the rectangle is added at the left-hand side of the gap.

The shortest neighbour policy places a new rectangle next to the gap-defining rectangle that ends at the lowest level, disregarding the actual height of this

rectangle. As the sheet side is infinitely long, it will never be considered the shortest neighbour. In case of a tie, this placement policy positions the rectangle at the left-hand side of the gap in all cases but one: if the gap is defined by both sheet sides, the rectangle is placed at the right-hand side of the gap. Although this definition of the shortest neighbour policy may appear somewhat complex, experiments show that it performs more efficiently than the inverse of the tallest neighbour policy. This is probably due to a stronger symmetry disruption compared to the other policies. Figure A.2 visualises the shortest neighbour policy. The gap in A.2 (a) is defined by both sheet sides, so the rectangle is placed at the right-hand side of the sheet. In A.2 (b), the rectangle fits exactly and completely fills the gap. In A.2 (c), the gap is defined by a rectangle on the left and the sheet side on the right, so the new rectangle is placed at the left-hand side. In A.2 (d) the rectangle fits exactly after rotation. In A.2 (e), the shortest neighbour is on the left-hand side of the gap, so that is where the new rectangle will be placed. Due to the tie in Figure A.2 (f), the rectangle is placed at the left-hand side of the gap.

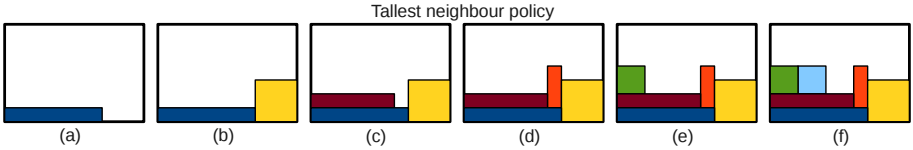

Figure A.1: Visual examples of the tallest neighbour policy.

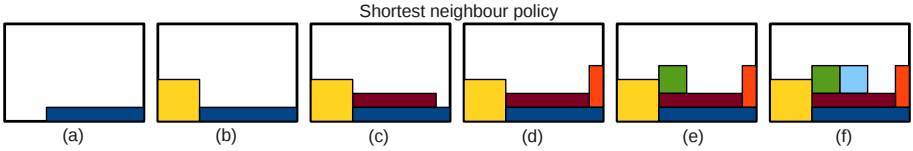

Figure A.2: Visual examples of the shortest neighbour policy.

A.1.3 New placement policies

Complementing the above placement policies we propose three new ones:

- Rightmost: place the fitting rectangle at the right-hand side of the gap
- MaxDiff: place the fitting rectangle so that the difference in top level with its neighbour is maximal

- MinDiff: place the fitting rectangle so that the difference in top level with its neighbour is minimal

The rightmost neighbour policy is the inverse of the leftmost neighbour policy. Burke et al. [2004] claim that using a rightmost neighbour policy would result in a mirror image of the leftmost neighbour policy, however this claim only holds for a few particular cases. While placing a rectangle at the right-hand side of the gap does indeed represent the inverse of placing it at the left-hand side, it constitutes mirroring only when no intermediate solution contains more than a single gap of equal level. Unless the gap location procedure for the rightmost neighbour policy is the inverse of the gap location procedure of the leftmost neighbour policy, the resulting solutions will not constitute mirroring. Thus, including the rightmost neighbour policy when utilising the best-fit heuristic is worthwhile, as it may strengthen the solution quality by breaking some of the symmetry. Section A.3 demonstrates that adding this placement policy effectively improves the results of the best-fit heuristic.

The MaxDiff neighbour policy is similar to the tallest neighbour policy, but places the rectangle next to the neighbour with which the difference in top level is maximal. In the case of ties, this placement policy behaves like the tallest neighbour policy, i.e. it will always place the rectangle at the left-hand side of the gap. Figure A.3 visualises this placement policy. In Figure A.3 (a), the new rectangle is placed at the right-hand side of the gap as the right-hand side gap defining rectangle ends at a lower level, while the left-hand side gap defining rectangle ends at the same level as the new rectangle. The same situation occurs in (b), but the right-hand side gap defining rectangle ends higher than the new rectangle. In Figure A.3 (c) both top levels are the same, and in Figure A.3 (d) the absolute difference in top level with the new rectangle is equal at both sides. With a tie presenting itself in both cases, the rectangle is placed at the left-hand side of the gap.

The MinDiff neighbour policy works similarly to the shortest neighbour policy, but places the rectangle next to the neighbour whose top level difference is minimal. In case of ties, this placement policy behaves exactly like the shortest neighbour policy. Figure A.4 visualises this placement policy. In Figures A.4 (a) and (b), the new rectangle is placed at the right-hand side of the gap, because the rectangle to the right has the same top level as the new rectangle, while the rectangle to the left has a lower/higher top level. Figure A.4 (c) demonstrates a tie as both neighbours have the same top level, so the rectangle is placed at the left-hand side. Figure A.4 (d) also shows a tie. Since the absolute height difference is equal for both gap defining rectangles, this situation also leads to a left-hand side placement of the rectangle.

A.1.4 New orderings

The original best-fit heuristic uses a decreasing width ordering. We suggest the addition of two more orderings to the solution process: decreasing height order and decreasing surface order. Including both orderings ensures a significant disruption in the sequence of the rectangles when compared to the width ordering alone.

Given a list of n ships in the default configuration, the decreasing height order sorts the items by decreasing height. Next, each item's rotated configuration is added to the list, directly after its default configuration. As far as the small example is concerned, the rectangle ordering becomes 25x24, 40x16 and 16x13, while the resulting list for the best-fit heuristic is 25x24, 24x25, 40x16, 16x40, 16x13 and 13x16. The new ordering leads to better results when, for example, some rectangles have a dimension larger than the sheet width. These items cannot be rotated and the best-fit heuristic often adds them towards the end of the procedure, leading to the formation of irremovable towers that detrimentally affect solution quality. By using the decreasing height order, those rectangles are added at the beginning of the search process, reducing the risk of generating towers. It should be noted that if the height ordering would be applied using the rotated configuration, all items would be placed in their rotated configuration (i.e. portrait), which is undesirable when trying to solve this type of problem.

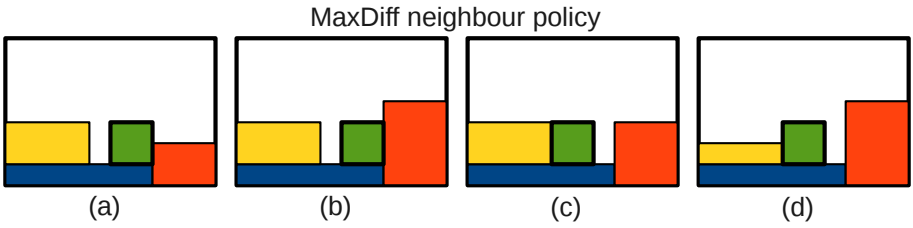

Figure A.3: A visual representation of the MaxDiff neighbour policy.

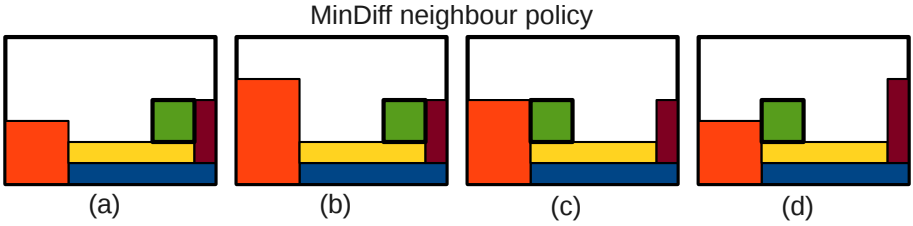

Figure A.4: A visual representation of the MinDiff neighbour policy.

Sorting by decreasing surface order results in a procedure where large rectangles are assigned the highest priority. The procedure for ordering the items is the same as that of the decreasing height ordering, resulting in the following item list: 40x16, 16x40, 25x24, 24x25, 16x13 and 13x16. The inclusion of these two additional orderings triples the computation time of the heuristic because the heuristic is run three times in stead of only once. Nevertheless the computation times remain short, even for large instances.

A.1.5 Three-way best-fit heuristic

By applying both the old and new placement policies and combining them with the decreasing width, height and surface orders, we create a high-performing extension to the best-fit heuristic. We call this new heuristic the three-way best-fit heuristic, as the rectangles are ordered in three different ways during the search for a good solution. In fact, this heuristic solves the same problem repeatedly, using a different ordering and placement policy combination each time. These orderings and placement strategies can easily be added or removed when necessary. For example, when all shapes under consideration are square, one of the proposed orders suffices because all the others will result in the same initial sequence. The pseudocode of the three-way best-fit heuristic is presented in Algorithm 5.

A.1.6 Special cases

When solving strip packing problems with the best-fit heuristic, particular problems may arise when dealing with problem instances that contain rectangles for which the largest dimension (called the rectangle's width, without loss of generality) is larger than the sheet width. The best-fit heuristic does not prioritise the placement of these items. Even worse, this kind of 'oversized' rectangle has a high probability of being among the last items that are placed. Since these rectangles cannot be placed on the stock sheet in landscape, they must be placed as a rotation candidate. Therefore, they will now be placed only if no other remaining item better fits the gap (Section A.1.1). To rephrase, the oversized rectangle will only be placed in the gap when none of the remaining items have one of their dimensions larger than the oversized rectangle's height and smaller than the gap. If the problem contains one rectangle of dimension 10x2 and six squares of dimension 3x3 on a sheet with width 9, the first (and largest) rectangle will be placed last. This will result in a very poor quality solution with total height of 16 (Figure A.5 (a)). A much better procedure is to place the oversized rectangle earlier on in the solution process, like in

Algorithm 5 Pseudo code of the three-way best-fit heuristic.

```
Input: sheetInfo S
Input: list L of rectangles
bestS ← empty Sheet
for Each ordering policy OrderPol do
    OrderedSeq ← all rectangles in L
    order OrderedSeq using OrderPol
    for Each placement policy do
        newS ← new Sheet with dimension from S
        RectangleSeq ← OrderedSeq
        while not all Rectangles added to newS do
            CurrGap ← Lowest Gap
            Get Best-Fitting Rectangle R
            if Best-Fitting Rectangle found then
                newS ← R at position in CurrGap determined by placement policy
                Remove R from RectangleSeq
                Update Skyline to reflect addition of R
            else
                Raise Gap to Lowest Neighbour
            end if
        end while
        Reduce towers (post-processing)
        if newS is better than bestS then
            bestS ← newS
        end if
    end for
end for
return bestS
```

Figure A.5 (b), where the total height is only 10. The impact of this special case characteristic can also be observed when applying the best-fit heuristic to the N9 instance [Burke et al., 2004] using the best-fit heuristic. This instance has two rectangles with a dimension larger than the sheet width. While one is placed on the sheet early in the solution process, the other is placed towards the end. This results in a solution containing a large tower that cannot be eliminated by rotation (Figure A.6). Therefore, one more rule is added to the three-way best-fit heuristic, namely not to use the default configuration of rectangles with a width larger than the sheet width. There are two ways to 'not use' the default configuration. A first approach is to create the ordered item list, and then remove the default configuration of all oversized rectangles. This does not change the rectangle placement nor the solution quality. A second approach is to create the ordered item list, and then replace the default configuration of the oversized rectangles by their rotated configuration. The experiments have shown that the second approach does indeed lead to better results when the instance contains oversized rectangles.

- Decreasing width: 10x2, 3x3, 3x3, ..., 3x3, 2x10

- First approach: 3x3, 3x3, ..., 3x3, 2x10
- Second approach: 2x10, 3x3, 3x3, ..., 3x3, 2x10

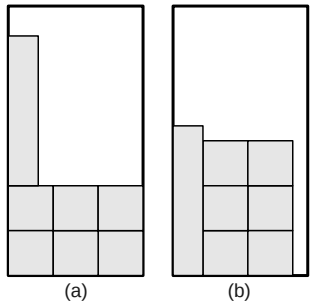

Figure A.5: A visual representation of the solution for the best-fit heuristic without the rotation rule (a) and with the rotation rule (b) for a test instance with one rectangle that has a dimension that is larger than the sheet width.

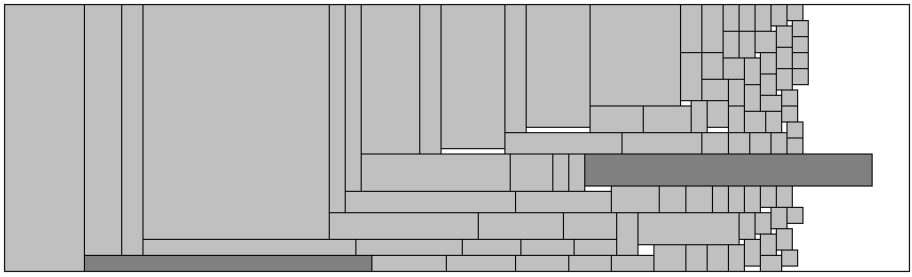

Figure A.6: A visual representation of the solution of the best-fit heuristic for the N9 instance from [Burke et al., 2004]. The rectangles for which the largest dimension is larger than the sheet width are coloured dark grey.

A.2 An optimal time three-way best-fit heuristic

Imahori and Yagiura [2010] analyse the time and space complexity of the original best-fit heuristic and introduce an optimal time best-fit heuristic. In this section, we present the optimal time data structures mentioned and discuss their applicability to the three-way best-fit heuristic.

A.2.1 Alternative data structures for the best-fit heuristic

Imahori and Yagiura [2010] store the sheet skyline using both a heap and a doubly linked list, allowing for a significant improvement in computation time when compared with the original data structures from Burke et al. [2004]. Each gap is now represented by an object with a start position, an end position and a height. The location and size of the lowest available gap is determinable in constant time, while the complexity of updating the skyline is reduced to $O(\log n)$. The original data structure requires $O(n)$ for locating the lowest available gap and constant time for updating the skyline.

The items are stored in a balanced binary tree, based on their width. Both the default and rotated configuration of each item are placed into this tree, thus enabling $O(\log n)$ complexity for locating the best-fitting rectangle for the current gap. This balanced binary tree structure contributes to another complexity reduction compared to the $O(n)$ of the original data structure.

A.2.2 Applicability of the data structures

There is no difference between the gap location and gap updating in the original best-fit heuristic and the three-way best-fit heuristic. Therefore, the optimal time gap location data structure can be used as is in the three-way best-fit heuristic.

The rectangle selection procedure, however, is not directly applicable due to the new item orderings. There is a mismatch between the alternative orderings of the items, which are based on the height/size of the rectangles, and the rectangle selection procedure, which is always based on the width of the gap. The height and surface orderings of the items fail to maintain consistency with respect to the width of items. It is therefore impossible to obtain the same rectangle placement sequence in $O(\log n)$ time as after an $O(n)$ linear search. We can, however, use the height/surface ordering while searching in a width-ordered balanced binary tree for the rectangle selection. This works as follows. The item's normal and rotated configurations are first stored in lists containing only other items with the same width (say w), while respecting their original height/size ordering. Each list is then linked to a 'dummy' item with that same width w, which is added to a decreasing width ordered binary tree. The balanced binary tree can be used in the same way as before, but instead of selecting the resulting dummy item, the first item in the associated list is selected for placement. This data structure yields a fitting rectangle in $O(\log n)$ time, as removing the first item from a list requires only constant time.

Nevertheless, the probability is high that a different rectangle will be selected than when using the original linear search.

We use the small example to elucidate the differences between the original linear search and the linked binary tree (Figure A.7). Applying the original rectangle selection procedure to a decreasing height ordered list, results in the following priority for placing the items: 25x24, 24x25, 40x16, 16x40, 16x13 and 13x16. The resulting *linked binary tree* is visualised in Figure A.7 (a). The white rectangles represent the dummy items stored in a balanced binary tree, while the coloured rectangles represent the actual items ('R' for the rotated configuration). Using the linear search, the resulting solution for a sheet width of 41 is presented in Figure A.7 (b). Applying the linked binary tree for the rectangle selection and the same height ordering, we obtain the result in Figure A.7 (c), which is clearly not equivalent to the result generated by the original procedure. Figure A.7 (c) also shows that the items with the largest height are not necessarily placed with the highest priority.

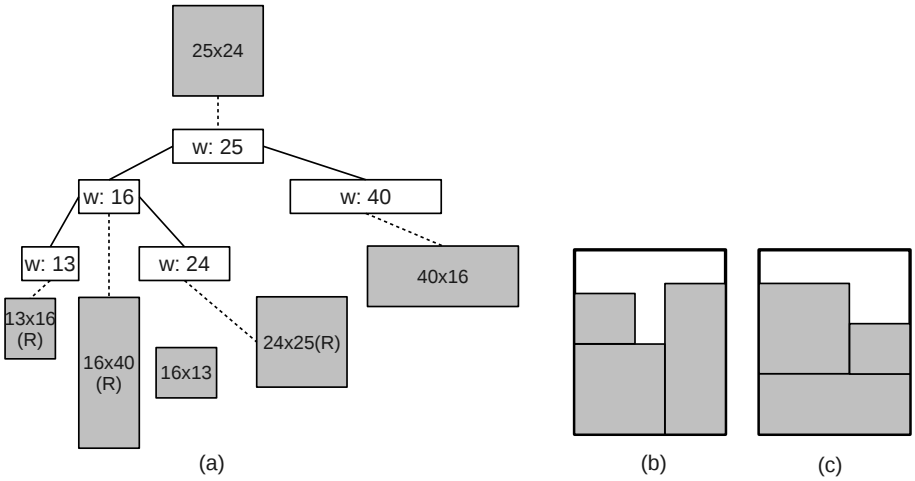

Figure A.7: Visualisation of the linked binary tree on the small example using a height ordering (a), and the resulting solution for the same problem using (b) the linear search, (c) the linked binary tree.

The main advantage of the height and surface based orderings is the disruption of the generated priority sequence in comparison to the decreasing width ordering. Applying the linked binary tree to these orderings, also results in a strong disruption of the priority sequence compared to the decreasing width ordering. Although the resulting sequence is different from the result of the linear search, the linked binary tree's complexity is advantageous. Compared to the $O(n^2)$

complexity of the original implementation, the computation time is much smaller for large problem instances. Even if the priority disruption between the different orderings is less effective than for the linear search, the possible decrease in solution quality will most likely be irrelevant given the strongly reduced computation times. It is therefore worthwhile to further investigate the heuristic's performance with the linked binary tree, especially for very large problem instances. We call this heuristic the three-way heuristic. The best-fit part has been removed from the name, given the fact that the rectangle obtained when using the height/area orderings is a *fitting* one, but not necessarily the *best-fitting* rectangle that would have been selected when using a linear search.

To conclude this section, we succinctly summarize the properties of the new heuristics. The three-way **best**-fit heuristic is based on the original rectangle selection with all three orderings, all six placement strategies and the optimal time gap location procedure. The **optimal time** three-way heuristic uses the same orderings and placement strategies, but implements both the optimal time gap location and the optimal time linked binary tree-based rectangle selection procedure. A final heuristic presented is the **combined** three-way heuristic, which solves each instance with both aforementioned heuristics and returns the best solution found.

A.3 Experiments

We discuss the performance of the three-way best-fit heuristic and its optimal time variant on a set of benchmark instances from the literature for which the optimal solution is known (Table A.1). Due to the very large computation times required to solve the $i19$ and $i20$ instances from Imahori and Yagiura [2010] with the original and three-way best-fit heuristic, these instances have only been addressed in terms of scalability (Section A.3.5). For all the other experiments, these very large instances were removed from the test set. First, we report on the improvements that can be made by adding the placement policies from Section A.1.3. After that, the effect of adding additional orderings is discussed, and we show the performance gain that can be obtained by combining both extensions into a three-way best-fit heuristic. Next, a comparison is made between several other best-fit improvements and a top performing metaheuristic for the strip packing problem. Finally, we report on the experiments concerning the scalability of the original best-fit heuristic [Burke et al., 2004], the optimal time best-fit heuristic [Imahori and Yagiura, 2010] and the three-way heuristics. All experiments were performed on a Dell Optiplex 755 with an Intel(R) Core(TM)2 Duo CPU E8600 (3.33GHz) and 8GB of memory running a 64-bit Scientific Linux. When referring to the optimality gap of a heuristic, the gap is

computed as follows: $100(Heuristic - Optimal)/Optimal(\%)$ in percent. The difference in solution quality between two heuristics is calculated using the following formula: $100(Heuristic1 - Heuristic2)/Heuristic1(\%)$.

Table A.1: Used benchmarks from the literature.

Data source	Problem category	#Problems	#Rectangles
Hopper [2000]	T1-T7, N1-N7	70	17 to 199
Hopper and Turton [2001]	C1-C7	21	16 to 197
Burke et al. [2004]	N1-N13	13	10 to 3152
Imahori and Yagiura [2010]	i4-i20	170	2^4 to 2^{20}

The results generated by the best-fit heuristic for the same data instances sometimes differ slightly between [Burke et al., 2004] and [Burke et al., 2009]. Solutions obtained by our best-fit implementation matched the results of either one of the two mentioned papers in all but four cases. Private communication enabled the clarification of this issue. Newly generated visual representations of the best-fit heuristic results provided by the authors of both papers matched our solutions exactly. After so many years, this mismatch can no longer be traced back to the original software. While the differences are negligible in most cases, for the N3 and N9 instances significant deviations exist between our solutions and those reported. Thus we decided to base the experimental comparison on our own implementation of the best-fit heuristic. The solution opted for, as such, also allows for a fair comparison between the best-fit heuristic and our improvements, since all results are based on the same best-fit implementation.

A.3.1 Placement policies

A summary of the effect of the three new placement policies on the solution quality is shown in Table A.2, presenting the increase in solution quality obtained when all placement policies are used compared to using the original best-fit heuristic. The experiments were performed using the decreasing width ordering. A table illustrating detailed results for all 267 instances and each individual placement strategy is available online [Verstichel, 2011]. The detailed results show that the performance of the placement policies is instance dependent, with certain policies performing better than average on some instances and poorer than average on others. Therefore, all placement policies should be combined for the best results. This combination performs statistically better than the original best-fit with more than 99.999% certainty. The average difference in solution quality is 0.27%. Whilst perhaps appearing to be a small increase in solution quality, it should be noted that the optimality gap is reduced by 5.14% on average, with a maximum of 50%.

The combination of all six placement policies increases the computation time, but instances with up to 16384 items remain solvable in less than one second when applying the optimal time heuristic. Instances with up to 4096 items can be solved in an equal amount of time when using the optimal gap location procedure, whereas the original implementation is limited to instances with just 512 items. The results also show that the rightmost policy indeed produces differing results compared with the leftmost policy in several instances. An analysis of these instances showed that they had at least one state in the solution process that contained multiple gaps at the same level. From here on, the solution construction was no longer symmetrical to that of the leftmost policy, proving true the statement made in Section A.1.3.

A.3.2 Orderings

In addition to the width ordering of the original best-fit heuristic, we proposed two more orderings: decreasing height ordering and decreasing surface ordering. A summary illustrating the effects on the solution quality of introducing these orderings is presented in Table A.2. The experiments were performed using the three placement strategies of the original best-fit heuristic. The detailed results [Verstichel, 2011] show that each ordering performs best on a number of instances, and worst on others. The results obtained combining all three orderings are significantly better than those obtained with the original best-fit heuristic with a certainty of 99.998%, while the average improvement is 0.14%. Furthermore, the optimality gap was reduced by up to 75%. However, applying the two additional orderings increases the computation time. Nevertheless, test instances with 16384 items are still solvable in less than one second using the optimal time heuristic. Using only the optimal gap location procedure enables solving instances with up to 2048 items in the same amount of time, whereas the original implementation is limited to 512 items.

A.3.3 Three-way best-fit

There exists further potential to improve the solution quality by integrating all ordering strategies and placement policies into a three-way best-fit heuristic. This three-way heuristic employs three different orderings (width, height and surface) and six different placement strategies (left, right, tallest, shortest, minDiff and maxDiff). Each instance is solved for all 18 ordering-placement combinations. Additionally we employ the rotation rule (Section A.1.6) to the heuristic to obtain the best results. The optimal time three-way heuristic, however, ignores this rule due to its different rectangle selection procedure. Table

A.2 shows a summary of the results for the three-way best-fit heuristic and optimal time three-way heuristic. The computation time increases compared to adding only the new placement policies or only the new orderings. Nevertheless, instances with over 1024 rectangles are still solvable in under one second. Using the three-way best-fit heuristic produces significantly better results compared to the previous two approaches. A T-test was applied for statistical analysis. The three-way best-fit heuristic performs better than the original best-fit heuristic with a confidence interval of more than 99.9999%. Furthermore, the average improvement of the three-way best-fit heuristic over the original is 0.47%, while the optimality gap is reduced by an average of 7.88%.

The optimal time three-way heuristic produces slightly worse results when compared to the three-way best-fit heuristic. For the largest instances however, (i14-i18) the optimal time three-way heuristic obtains better results than the three-way best-fit heuristic in several cases. A statistical analysis shows that the results of the two three-way heuristics are not significantly different ($p-value = 0.158$). When compared with the three-way best-fit heuristic, the merger of both heuristics performs better (Table A.2), alongside a negligible increase in computation time. When considering the largest instances with 2^{18} rectangles, we find that the optimality gap is reduced by 8.13% on average by combining both heuristics, compared to 3.21% when using the three-way best-fit heuristic. Simultaneously, the required computation time is increased by, at most, 0.6%.

A.3.4 Comparison to state of the art (meta)heuristics

Two other enhancements to the best-fit heuristic have been presented in the literature: the bidirectional best-fit heuristic [Aşık and Özcan, 2009] and the simulated annealing enhancement of the best-fit heuristic [Burke et al., 2009]. We will compare the enhancements mentioned and the top performing GRASP approach for the strip packing problem [Alvarez-Valdes et al., 2008] to the three-way best-fit heuristic. In Table A.3, the results of the four heuristics for the Hopper and Turton [2001] and Burke et al. [2004] instances are shown. The results were taken from the papers cited and we refer the reader to those papers for further information. When considering solution quality, the GRASP approach is clearly superior. The three-way approach is able to compete with the other two best-fit based heuristics on several instances, while not producing the best results. With respect to the computation times, however, the three-way heuristics are clearly superior, requiring only a fraction of the computation time required by any of the other heuristics. When compared to the original best-fit heuristic, both three-way approaches have similar or lower computation times, while obtaining significantly better results.

Table A.2: Average and maximal improvements of the total required height ('Solution quality'), average and maximal decreases of the optimality gap ('Optimality gap'), and number of improved instances ('#') compared to the original best-fit heuristic, for each test setting.

All placement policies					
	Solution quality		Optimality gap		
Test set	%average	%max	%average	%max	#
Hopper	0.48%	4.96%	7.25%	44.44%	21/70
Hopper Turton	0.31%	4.55%	5.16%	50.00%	3/21
Burke	0.17%	0.95%	6.67%	33.33%	3/13
Imahori	0.17%	3.58%	4.02%	46.32%	50/150
All	0.27%	4.96%	5.14%	50.00%	77/254
All orderings					
	Solution quality		Optimality gap		
Test set	%average	%max	%average	%max	#
Hopper	0.06%	2.27%	0.80%	25.00%	3/70
Hopper Turton	0.78%	12.50%	6.75%	75.00%	3/21
Burke	0.60%	5.52%	6.86%	69.23%	2/13
Imahori	0.04%	1.23%	2.07%	27.70%	22/150
All	0.14%	12.50%	2.35%	75.00%	30/254
Three-way best-fit heuristic					
	Solution quality		Optimality gap		
Test set	%average	%max	%average	%max	#
Hopper	0.64%	5.38%	9.09%	52.17%	25/70
Hopper Turton	1.36%	12.50%	14.29%	75.00%	7/21
Burke	1.13%	5.52%	18.48%	69.23%	7/13
Imahori	0.21%	3.58%	5.49%	46.32%	61/150
All	0.47%	12.50%	7.88%	75.00%	100/254
Optimal time three-way heuristic					
	Solution quality		Optimality gap		
Test set	%average	%max	%average	%max	#
Hopper	0.48%	4.96%	7.25%	44.44%	21/70
Hopper Turton	0.31%	4.55%	5.16%	50.00%	3/21
Burke	0.17%	0.95%	6.67%	33.33%	3/13
Imahori	0.17%	3.58%	4.87%	46.32%	54/150
All	0.27%	4.96%	5.64%	50.00%	81/254
Combined three-way heuristics					
	Solution quality		Optimality gap		
Test set	%average	%max	%average	%max	#
Hopper	0.64%	5.38%	9.09%	52.17%	25/70
Hopper Turton	1.36%	12.50%	14.29%	75.00%	7/21
Burke	1.13%	5.52%	18.48%	69.23%	7/13
Imahori	0.21%	3.58%	6.13%	46.32%	64/150
All	0.47%	12.50%	8.25%	75.00%	103/254

Table A.3: Comparison of the best-fit heuristic, two best-fit based (meta)heuristics [Aşık and Özcan, 2009; Burke et al., 2009], a top performing GRASP approach [Alvarez-Valdes et al., 2008] and both three-way heuristics.

			Best-Fit (BF)		Bilinear BF		SA Best-Fit		GRASP		Three-way BF		Optimal time three-way	
Label	Items	Optimal	Result	Time (s)	Result	Time (s)	Result	Time (s)	Result	Time (s)	Result	Time (s)	Result	Time (s)
N1	10	40	45	< 0.01	40	0.03	40	∼60.00	40	∼60.00	44	< 0.01	45	< 0.01
N2	20	50	53	< 0.01	52	0.02	50	∼60.00	50	∼60.00	53	< 0.01	53	< 0.01
N3	30	50	54	< 0.01	52	0.05	51	∼60.00	51	∼60.00	52	< 0.01	54	< 0.01
N4	40	80	86	< 0.01	82	0.09	82	∼60.00	81	∼60.00	86	< 0.01	86	< 0.01
N5	50	100	105	< 0.01	104	0.14	103	∼60.00	102	∼60.00	104	< 0.01	104	< 0.01
N6	60	100	102	< 0.01	102	0.16	102	∼60.00	101	∼60.00	102	< 0.01	102	< 0.01
N7	70	100	107	< 0.01	106	0.22	104	∼60.00	101	∼60.00	106	< 0.01	107	< 0.01
N8	80	80	83	< 0.01	82	0.28	82	∼60.00	81	∼60.00	83	< 0.01	83	< 0.01
N9	100	150	163	< 0.01	152	0.33	152	∼60.00	151	∼60.00	154	< 0.01	163	< 0.01
N10	200	150	153	< 0.01	151	1.13	152	∼60.00	151	∼60.00	152	0.01	152	0.01
N11	300	150	153	< 0.01	151	2.14	153	∼60.00	151	∼60.00	152	0.009	152	0.017
N12	500	300	305	< 0.01	303	5.95	306	∼60.00	303.2	∼60.00	305	0.013	305	0.025
N13	3152	960	964	0.34	964	163.42	964	∼60.00	963	∼60.00	964	0.586	964	0.235
C1-P1	16	20	21	< 0.01	20	0.02	20	∼60.00	20	∼60.00	21	< 0.01	21	< 0.01
C1-P2	17	20	22	< 0.01	21	0.02	20	∼60.00	20	∼60.00	21	< 0.01	21	< 0.01
C1-P3	16	20	24	< 0.01	21	0.02	20	∼60.00	20	∼60.00	21	< 0.01	24	< 0.01
C2-P1	25	15	16	< 0.01	16	0.05	16	∼60.00	15	∼60.00	16	< 0.01	16	< 0.01
C2-P2	25	15	16	< 0.01	16	0.03	16	∼60.00	15	∼60.00	16	< 0.01	16	< 0.01
C2-P3	25	15	16	< 0.01	15	0.03	16	∼60.00	15	∼60.00	16	< 0.01	16	< 0.01
C3-P1	28	30	32	< 0.01	30	0.05	31	∼60.00	30	∼60.00	32	< 0.01	32	< 0.01
C3-P2	29	30	34	< 0.01	33	0.05	31	∼60.00	31	∼60.00	32	< 0.01	34	< 0.01
C3-P3	28	30	33	< 0.01	31	0.05	31	∼60.00	30	∼60.00	32	< 0.01	33	< 0.01
C4-P1	49	60	63	< 0.01	62	0.11	61	∼60.00	61	∼60.00	63	< 0.01	63	< 0.01
C4-P2	49	60	62	< 0.01	62	0.11	61	∼60.00	61	∼60.00	62	< 0.01	62	< 0.01
C4-P3	49	60	62	< 0.01	61	0.11	61	∼60.00	61	∼60.00	62	< 0.01	62	< 0.01
C5-P1	72	90	93	< 0.01	91	0.19	91	∼60.00	91	∼60.00	92	< 0.01	92	< 0.01
C5-P2	73	90	92	< 0.01	92	0.19	91	∼60.00	91	∼60.00	92	< 0.01	92	< 0.01
C5-P3	72	90	93	< 0.01	91	0.19	92	∼60.00	91	∼60.00	93	< 0.01	93	< 0.01
C6-P1	97	120	123	< 0.01	122	0.33	122	∼60.00	121.9	∼60.00	123	< 0.01	123	< 0.01
C6-P2	97	120	122	< 0.01	121	0.31	121	∼60.00	121.9	∼60.00	122	< 0.01	122	< 0.01
C6-P3	97	120	124	< 0.01	122	0.38	122	∼60.00	121.9	∼60.00	123	< 0.01	123	< 0.01
C7-P1	196	240	246	0.01	243	1.2	244	∼60.00	244	∼60.00	244	< 0.01	246	< 0.01
C7-P2	197	240	244	< 0.01	244	1.05	244	∼60.00	242.9	∼60.00	244	< 0.01	244	< 0.01
C7-P3	196	240	245	< 0.01	244	1.14	245	∼60.00	243	∼60.00	245	< 0.01	245	< 0.01
Total	**6065**	**4035**	**4156**	**0.403**	**4099**	**179.59**	**4097**	**∼2040**	**4073.8**	**∼2040**	**4129**	**0.67**	**4150**	**0.376**

A.3.5 Scalability

Imahori and Yagiura [2010] introduced a large test set for the orthogonal strip packing problem. The test set contains instances with up to 2^{20} rectangles, and enables easy comparison of the different heuristics' scalability. To make a fair analysis, we first compare the run times of the three implementations using decreasing width ordering and the left placement policy, i.e. each heuristic constructs exactly one solution. Figure A.8 illustrates that the original implementation [Burke et al., 2004] is slower for all instances. The three-way best-fit heuristic clearly benefits from the optimal gap location process, as the computation times are strongly reduced when compared to the original implementation. However, this effect decreases as the instance size increases. The optimal time best-fit heuristic [Imahori and Yagiura, 2010] performs similarly to the optimized implementation for instances with less than 1024 pieces. For all the larger instances, the optimal time implementation is significantly faster due to its efficient rectangle selection procedure.

When comparing the original best-fit heuristic to the three-way best-fit heuristic and the optimal time three-way heuristic, the same trend becomes visible. Despite the fact that the 'three-way' heuristics solve the same problem 18 times, which is six times more than the original best-fit heuristic, they maintain comparable or smaller computation times. The original best-fit is slightly faster only for the smallest of data instances, but the computation times still remain under 0.01 seconds. Figure A.9 shows the computation times for the three heuristics. The computation times are scaled logarithmically, and each instance contains twice as many rectangles as the preceding one. Optimization of the gap location process enables the three-way best-fit heuristic to solve all but the largest instances in comparable time to the original best-fit heuristic. In addition, the optimal time implementation [Imahori and Yagiura, 2010] makes the heuristic significantly faster for all but the smallest test instances. For instances with 2^{18} items, the optimal time three-way heuristic requires only 1.60% of the time needed by the original best-fit heuristic and 0.46% of the time needed by the three-way best-fit heuristic. While both the best-fit heuristic and three-way best-fit heuristic can solve instances with up to 1024 rectangles in under a second, the optimal time three-way heuristic can solve instances with up to 8192 rectangles in the same amount of time.

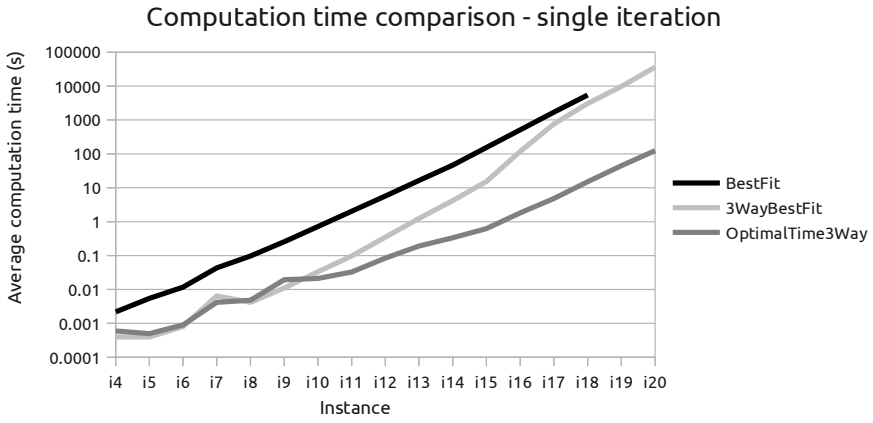

Figure A.8: Average computation times of the original best-fit, three-way best-fit, and optimal time three-way best-fit, when constructing only one solution (width-ordering and leftmost policy) for the Imahori and Yagiura instances.

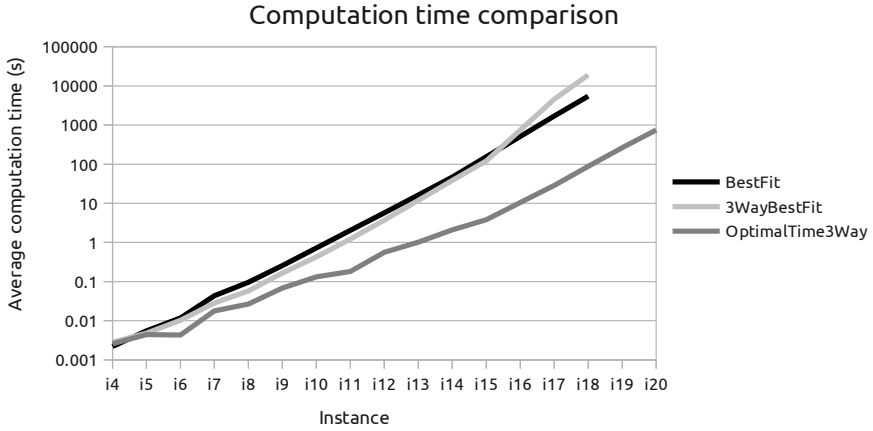

Figure A.9: Average computation times of the original best-fit, three-way best-fit and optimal time three-way heuristic, for the Imahori and Yagiura instances.

A.4 Conclusion

We have presented an improved heuristic for the orthogonal strip packing problem, based on the best-fit heuristic from Burke et al. [2004]. We extended this best-fit heuristic with additional item orderings and the introduction of new placement policies. The additional orderings allow for a larger diversification of the search by disrupting the rectangle sequence, while the new placement policies break the symmetry between different solutions by varying the placement rules. This new heuristic performs significantly better than the original best-fit heuristic on a large test set from the literature. The addition of the new placement policies and orderings increased the computation time of the heuristic. Thus we undertook development of a faster implementation of the heuristic. The three-way best-fit heuristic stores and locates the gaps in a more efficient way [Imahori and Yagiura, 2010] to reduce its computational complexity. Due to this improvement, the three-way best-fit heuristic has smaller computation times than the original best-fit heuristic for all but the largest problem instances. In addition, we introduced an optimal time variant of the three-way best-fit heuristic. This optimal time three-way heuristic is based on the optimal time best-fit heuristic [Imahori and Yagiura, 2010]. The results achieved are slightly different from those generated by the three-way best-fit heuristic, as the rectangle selection procedure produces different rectangle sequences for the new item orderings. There is no significant difference in solution quality between both three-way heuristics. The optimal time three-way heuristic is, however, significantly faster than the three-way best-fit heuristic on all but the smallest instances. When the quality of the solutions is more important than the computation times, combined usage of both three-way heuristics is advised for all but the largest problems. When more than 2^{16} items need to be placed, the optimal time three-way heuristic is more than 100 times faster than the three-way best-fit heuristic. When comparing the new three-way heuristics with other improvements to the best-fit heuristic and a top performing metaheuristic, we find that the new heuristics are significantly faster than those approaches. Even for small instances, our approach requires only a fraction of the computation time required by the other solution methods.

Appendix B

MOGLi: User interface

This appendix presents MOGLi's design and configuration options. With the solution approaches from Chapter 3 at its foundations this tool offers proven solution quality and responsiveness while enabling the user to alter solutions by means of an intuitive user interface. We first show the initial screen where traffic can be reviewed and edited. Next the solution visualisation screen is discussed, along with the available user interactions for editing a generated solution. The appendix ends with a description of the settings screen where all important parameters can be set in an intuitive way.

B.1 Traffic screen

Figure B.1 shows the traffic screen for a list of fourteen arriving ships. This screen contains all ship information, such as the ship names, dimensions, tugboat requirements, arrival times and types. Furthermore, the current chamber can be selected here, along with the lockage's index, planned execution time and direction. When a chamber has different length configurations, the available chamber length can also be selected here. New ships can be added to the list manually or by copying the ship's properties from another system, and ship properties can be updated at any time. Closing a lockage before reaching full chamber capacity is very straightforward, as is the manual addition of a ship to a lockage (Figure B.2). The arrow buttons allow lock masters to make changes in ship priority (Figure B.3), while cancelled ships can be removed from the list using the waste bin button (Figure B.4).

B.2 Solution screen

Given the current ship list, a (set of) solution(s) is generated after pressing the solve button (Figure B.5). The screen contains all necessary information about

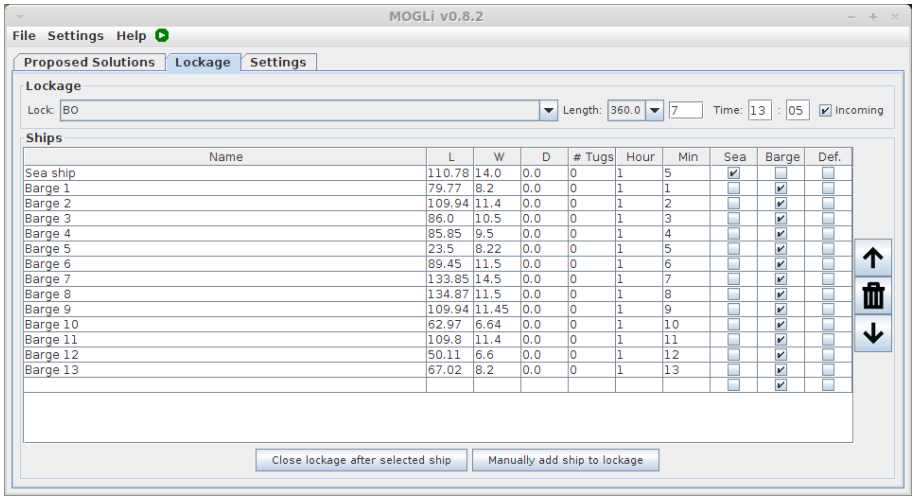

Figure B.1: The traffic screen of MOGLi containing the information of fourteen arriving ships.

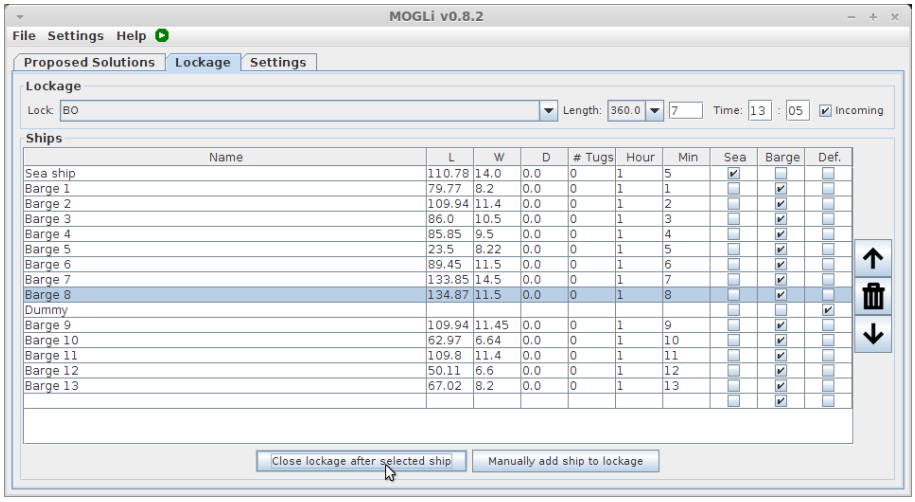

Figure B.2: Force-closing a lockage in MOGLi.

the current lockage such as the selected chamber and its dimensions, the travel direction and lockage index. The position of the seagoing vessels (green) and barges (grey) in the chamber is displayed for each suggested solution along with the total number of ships transferred, the solution method and the required computation time. The water colour in the lock depends on the travel direction,

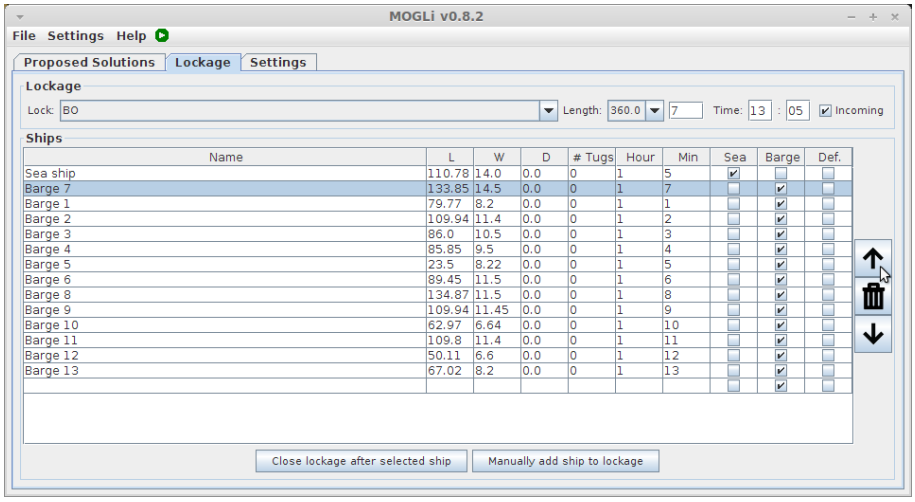

Figure B.3: Changing a ship's priority in MOGLi.

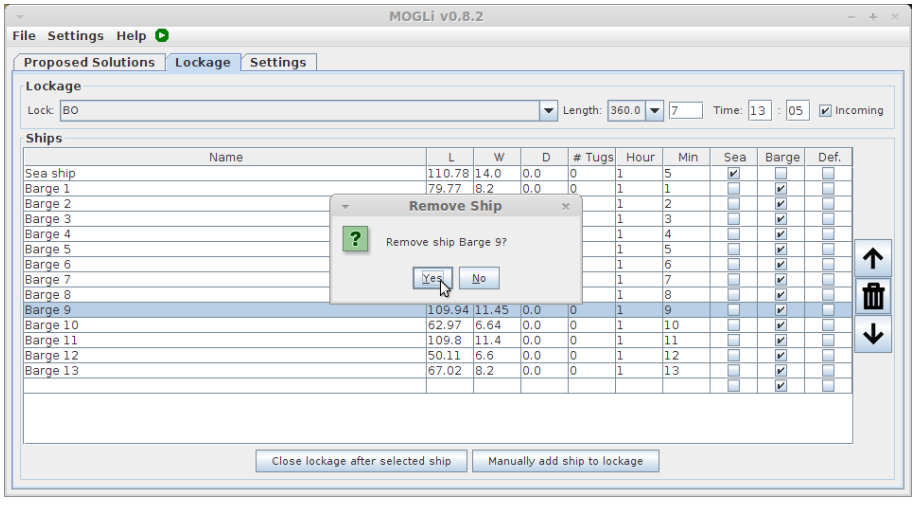

Figure B.4: Removing a ship from the traffic list in MOGLi.

with light blue indicating an upstream (or inbound) lockage and dark blue a downstream (or outbound) one. Hovering over a ship displays its properties, including the ship's name, tugboat requirements, arrival time, dimensions and minimal safety distances (Figure B.6), while hovering over free space quantifies the remaining lateral distance between the adjoining ships (Figure B.7). A ship can be moved to a different position with the drag functionality that includes a toggle-able snap function with real-time indication of the ship's destination position (Figure B.8). Figure B.9 shows how ships placed in violation with one of the mooring or overlap constraints are coloured red.

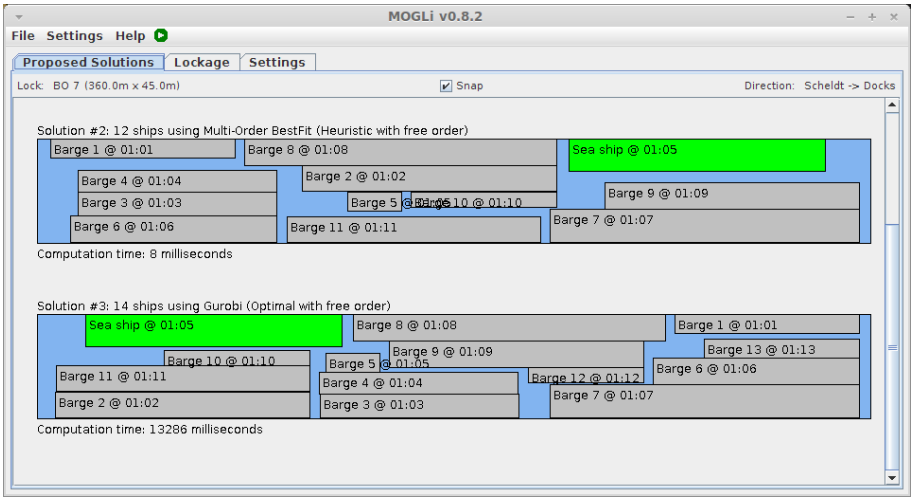

Figure B.5: The solution screen of MOGLi, showing two out of three suggested solutions.

B.3 Settings screen

The settings screen allows the user to edit all important parameters influencing the ship placement process. In the top part of the settings screen (Figure B.10) the user can select which algorithms will be applied to the problem at hand. Each selected solver will report a possible way of positioning the current ships in the selected chamber. Some high level parameters of the solvers, such as the used orderings (Figure B.11) or computation time limit (Figure B.12) are also configurable.

The bottom part of the settings screen provides the user with ample opportunity for personalizing the minimal safety distance requirements for each individual

lock. This is an important feature: the safety distances depend on the chamber and on the ship sizes and types. Moreover, the preferred margins may differ between lock masters: some transfer three adjacent 11.4 meter wide ships in a 35 meter wide chamber, leaving a total margin of 0.8 meters, while others transfer at most two of these ships side by side under the same circumstances.

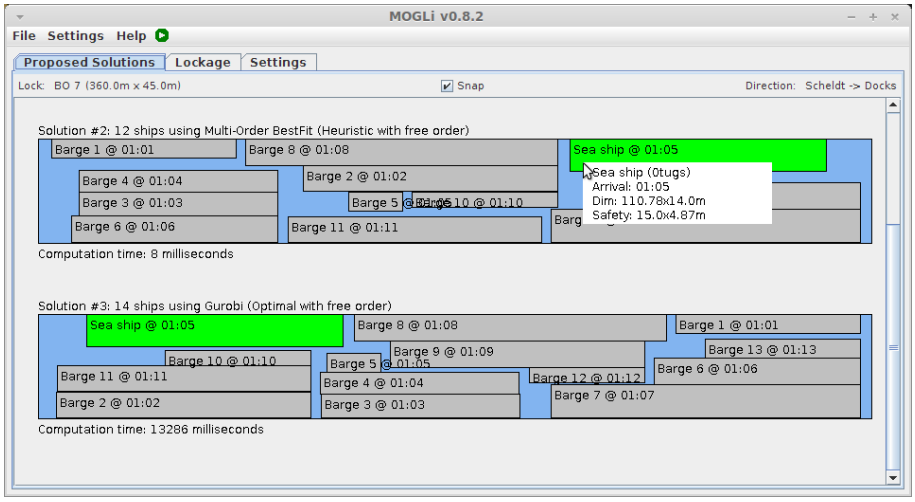

Figure B.6: Hovering over a ship displays additional information.

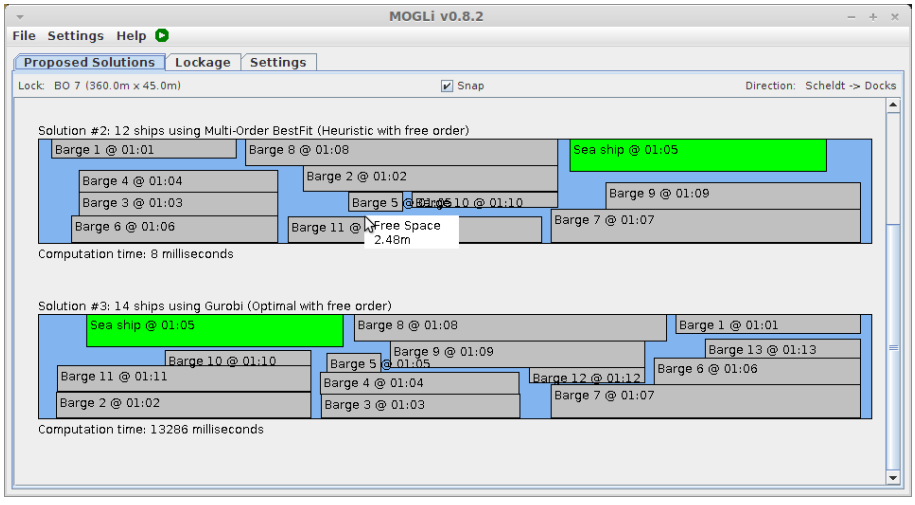

Figure B.7: Hovering over free space quantifies the remaining lateral distance.

Seagoing vessels are divided into classes based on their dimensions as shown in Figure B.10. Different safety distance settings must be applied for each class. Figure B.11 shows the options for barges, where a single set of safety distances

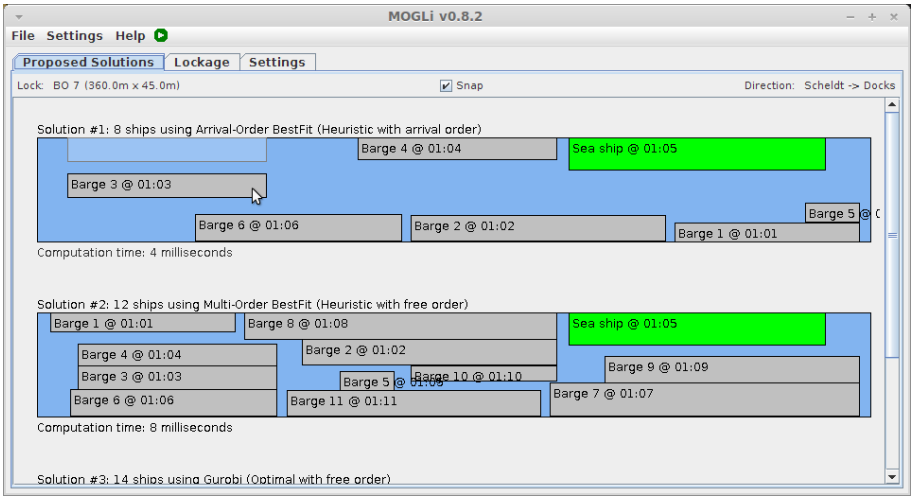

Figure B.8: Real-time indication of a dragged ship's destination position with the snap-function.

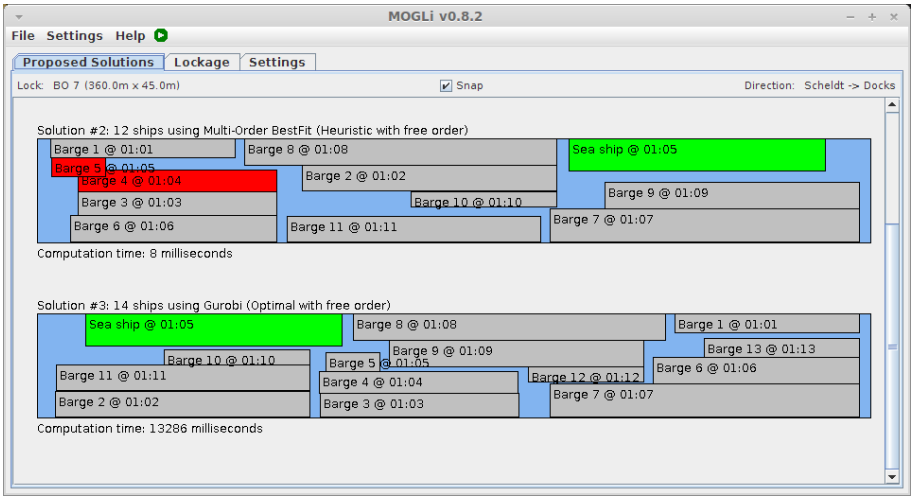

Figure B.9: Ships that violate the overlapping or mooring constraints are coloured red.

may be defined. The lateral safety distance for barges is a cumulative one. Enforcing a cumulative margin ensures that the last ship entering a row doesn't get clamped between its neighbouring ship(s) and/or the quay. The corridor constraint settings for tugboats are visualized in Figure B.12.

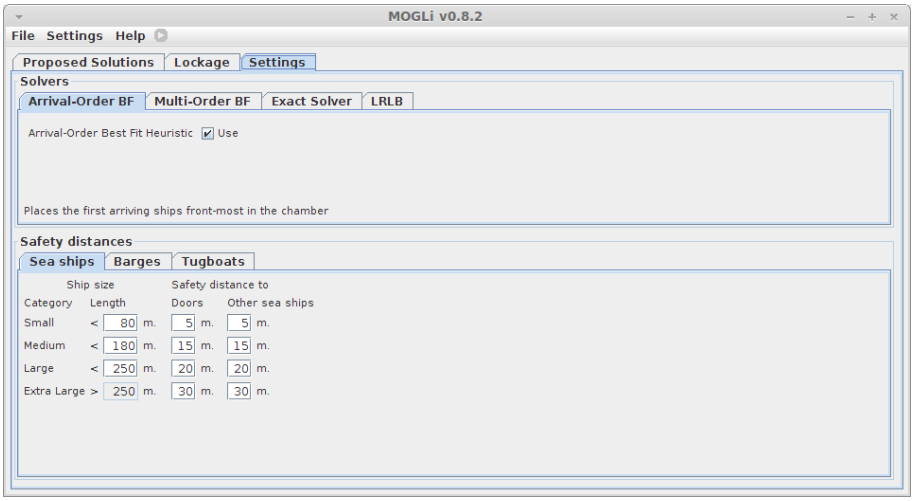

Figure B.10: The settings screen of MOGLi displaying the Arrival-Order Best Fit and Seagoing vessel safety distances tabs.

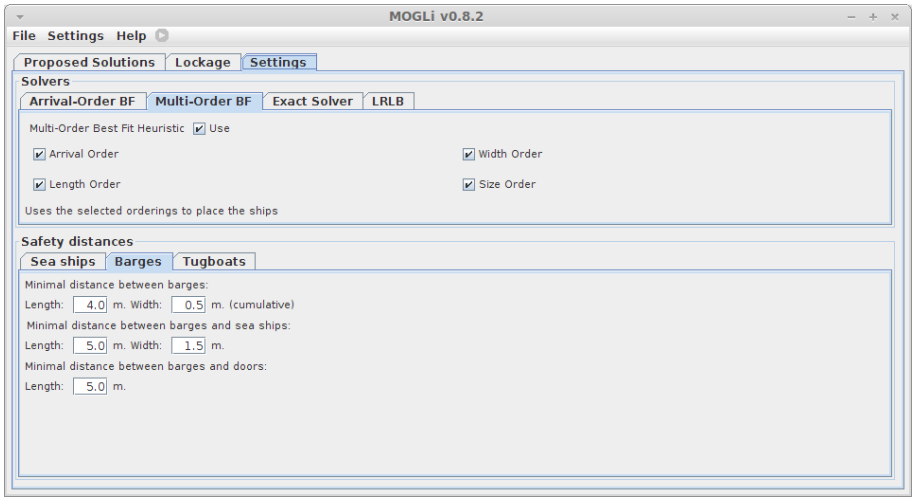

Figure B.11: The settings screen of MOGLi displaying the Multi-Order Best Fit and Barge safety distances tabs.

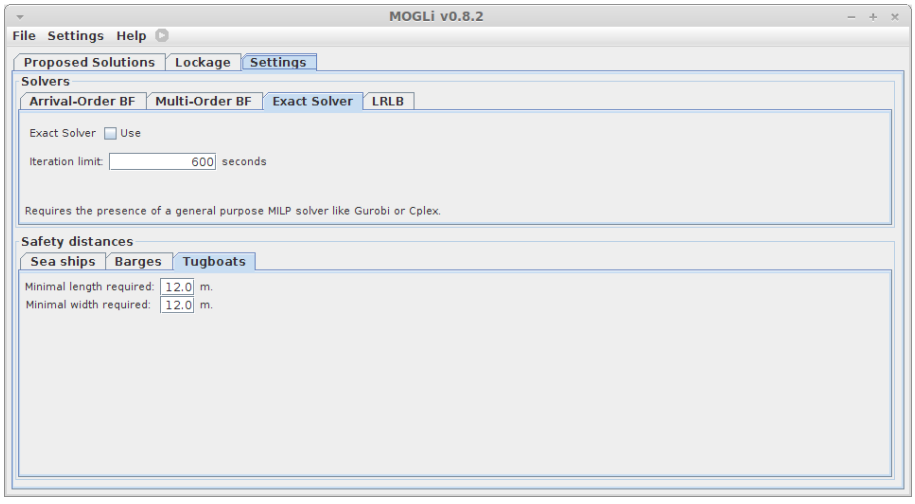

Figure B.12: The settings screen of MOGLi displaying the Exact approach and Tugboats tabs.

Bibliography

Ö. Aşık and E. Özcan. Bidirectional best-fit heuristic for orthogonal rectangular strip packing. *Annals of Operations Research*, 172(1):405–427, 2009.

R. Alvarez-Valdes, F. Parreno, and J. Tamarit. Reactive grasp for the strip-packing problem. *Computers & Operations Research*, 35(4):1065–1083, 2008.

Lihui Bai and Paul A. Rubin. Combinatorial Benders Cuts for the Minimum Tollbooth Problem. *Operations Research*, 57(6):1510–1522, November 2009.

B.S. Baker, E.G. Coffman_jr, and R.L. Rivest. Orthogonal Packings in Two Dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

N. Balakrishnan, J.J. Kanet, and V. Sridharan. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, 26(2):127–141, 1999.

J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238 – 252, 1962.

E.K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52:655 – 671, 2004.

E.K. Burke, G. Kendall, and G. Whitwell. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. *INFORMS Journal on Computing*, 21(3):505–516, 2009.

B. Chazelle. The Bottomn-Left Bin-Packing Heuristic: An Efficient Implementation. *IEEE Transactions on Computers*, C-32(8):697–707, 1983.

G. Codato and M. Fischetti. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

S. Coene and F.C.R. Spieksma. The lockmaster's problem. Technical report, KU Leuven, https://lirias.kuleuven.be/handle/123456789/318524, 2011.

Jean-François Côté, Mauro Dell'Amico, and Manuel Iori. Combinatorial benders' cuts for the strip packing problem. Technical Report CIRRELT-2013-27, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation and Department of Computer Science and Operations Research, Université de Mon (CIRELT), April 2013.

H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

European Commission. Communication from the commission: A sustainable future for transport - towards an integrated, technology-led and user friendly systems. `http://ec.europa.eu/transport/media/publications/doc/2009_future_of_transport_en.pdf`, 2009.

European Commission. Transport white paper: roadmap to a single european transport area – towards a competitive and resource efficient transport system. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2011:0144:FIN:EN:PDF`, 2011.

M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman & Co Ltd, first edition edition, 1979.

A.M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237 – 260, 1972.

J.N. Hooker and G. Ottoson. Logic-based benders decomposition. *Mathematical Programming*, 96:33 – 60, 2003.

E. Hopper. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. PhD thesis, Cardiff University, UK, 2000.

E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128:34–57, 2001.

S. Imahori and M. Yagiura. The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio. *Computers & Operations Research*, 37(2):325–333, 2010.

S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165–181, 1996.

J.Y-T. Leung, T.W. Tam, C.S. Wong, G.H. Young, and F.Y.L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3): 271 – 275, 1990.

R.M. Nauss. Optimal sequencing in the presence of setup times for tow/barge traffic through a river lock. *European Journal of Operational Research*, 187: 1268 – 1281, 2008.

T.E. Notteboom and J-P. Rodrigue. Port regionalization: towards a new phase in port development. *Maritime Policy & Management*, 32(3):297–313, 2005.

nv De Scheepvaart. Annual report 2012 (in dutch). `http://www.descheepvaart.be/uploads/scheepvaart/FILE_1DAC7D33-98D9-4971-978F-12C499037150.PDF`, 2012.

D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167, 2005.

Port of Antwerp. Annual report 2012. `http://www.portofantwerp.com/en/annual-report-2012`, 2012.

R. Rasmussen and M. Trick. A benders approach for the constrained minimum break problem. *European Journal of Operational Research*, 177(1):198–213, 2007.

L.D. Smith, D.C. Sweeney, and J.F. Campbell. Simulation of alternative approaches to relieving congestion at locks in a river transportion system. *Journal of the Operational Research Society*, 60(4):519–533, 2009.

L.D. Smith, R.M. Nauss, D.C. Mattfeld, J. Li, J.F. Ehmke, and M. Reindl. Scheduling operations at system choke points with sequence-dependent delays and processing times. *Transportation Research Part E: Logistics and Transportation Review*, pages 669–680, 2011.

R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30:1–52, 2008.

Tony T. Tran and J. Christopher Beck. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 774–779. IOS Press, 2012.

J. Verstichel. Website with the detailed results of the three-way heuristic. http://allserv.kahosl.be/∼jannes/stockcutting, 2011.

J. Verstichel. Project web page of lock scheduling with parallel chambers. `http://allserv.kahosl.be/~jannes/lockplanning`, 2012.

J. Verstichel and G. Vanden Berghe. A late acceptance algorithm for the lock scheduling problem. *Logistik Management*, pages 457 – 478, 2009.

J. Verstichel, P. De Causmaecker, F.C.R. Spieksma, and G. Vanden Berghe. Exact and heuristic methods for placing ships in locks. *European Journal of Operational Research*, 2013a. doi: 10.1016/j.ejor.2013.06.045. In Press.

J. Verstichel, P. De Causmaecker, F.C.R. Spieksma, and G. Vanden Berghe. The generalized lock scheduling problem: An exact approach. Technical report, 2013b.

J. Verstichel, P. De Causmaecker, and G. Vanden Berghe. An improved best fit heuristic for the orthogonal strip packing problem. *International Transactions in Operational Research*, 20:711–730, 2013c.

Walloon Government. The Strépy-Thieu boat lift. `http://services-techniques.met.wallonie.be/en/waterways/strepythieu_boat_lift/`, 2013a.

Walloon Government. Full and summarized year reports 2003-2011 of the Wallonian inland waterways (in French). `http://voies-hydrauliques.wallonie.be/opencms/opencms/fr/nav/navstat/docstat.html`, 2013b.

X. Wang, Y. Zhao, P. Sun, and X. Wang. An analysis on convergence of data-driven approach to ship lock scheduling. *Mathematics and Computers in Simulation*, 88(0):31 – 38, 2013.

G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109 – 1130, 2007.

H.G. Wilson. On the applicability of queueing theory to lock capacity analysis. *Transportation Research*, 12:175 – 180, 1978.

X. Zhang, X. Yuan, and Y. Yuan. Improved hybrid simulated annealing algorithm for navigation scheduling for the two dams of the three gorges project. *Computers & Mathematics with Applications*, 56(1):151 – 159, 2008.

Curriculum

April 23, 1985	Born in Oudenaarde, Belgium
2004-2007	Bachelor of Science in Industrial Engineering: Electronics-ICT, KAHO Sint-Lieven, Gent, Belgium
2007-2008	Master of Science in Industrial Engineering: Electronics-ICT, KAHO Sint-Lieven, Gent, Belgium
2008-present	Research associate at the Department of Industrial Engineering, Computer Science, KAHO Sint-Lieven, Gent, Belgium
2010-present	Doctoral student under IWT grants 091152 and 093152, Computer Science, KU Leuven campus Kortrijk, Kortrijk, Belgium

List of publications

Articles in internationally reviewed academic journals

- Verstichel, J., De Causmaecker, P., Spieksma, F., Vanden Berghe, G. (2013). Exact and heuristic methods for placing ships in locks. European Journal of Operational Research. *doi:10.1016/j.ejor.2013.06.045*
- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2013). An improved best fit heuristic for the orthogonal strip packing problem. International Transactions in Operational Research, 20 (5), 711-730.
- Wauters, T., Verstichel, J., Vanden Berghe, G. (2013). An effective shaking procedure for 2D and 3D strip packing problems. Computers & Operations Research, 40 (11), 2662-2669.
- Vancroonenburg, W., Verstichel, J., Tavernier, K., Vanden Berghe, G. (to appear). Automatic air cargo selection and weight balancing: a mixed integer programming approach. Transportation Research E, Logistics and Transportation Review.

Papers at international scientific conferences and symposia, published in full in proceedings

- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2011). Scheduling algorithms for the lock scheduling problem. In Zak, J. (Ed.), Procedia - Social and Behavioral Sciences: Vol. 20. The State of the Art in the European Quantitative Oriented Transportation and Logistics Research - 14th Euro Working Group on Transportation & 26th Mini Euro Conference & 1st European Scientific Conference on Air Transport. Poznan, Poland, 6-9 September 2011 (pp. 806-815) ELSEVIER.

- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2011). Enhancements to the best fit heuristic for the orthogonal stock-cutting problem. Proceedings of the VII ALIO–EURO – Workshop on Applied Combinatorial Optimization. ALIO/EURO Workshop on Applied Combinatorial Optimization. Porto, Portugal, 4-6 May 2011 (pp. 112-115).

- Verstichel, J., Vancroonenburg, W., Souffriau, W., Vanden Berghe, G. (2011). A mixed integer programming approach to the aircraft weight and balance problem. In Zak, J. (Ed.), Procedia - Social and Behavioral Sciences: Vol. 20. The State of the Art in the European Quantitative Oriented Transportation and Logistics Research – 14th Euro Working Group on Transportation & 26th Mini Euro Conference & 1st European Scientific Conference on Air Transport. Poznan, Poland, 6-9 September 2011 (pp. 1051-1059) ELSEVIER.

- Verstichel, J., Vanden Berghe, G. (2009). A late acceptance algorithm for the lock scheduling problem. In: Voss S., Pahl J., Schwarze S. (Eds.), Logistik Management. Heidelberg: Springer, 457-478.

Meeting abstracts, presented at international scientific conferences and symposia, published or not published in proceedings or journals

- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2013). Placing ships in locks: a decision support approach using exact and heuristic methods. 10th ESICUP Meeting. Lille, France, 24-26 April 2013.

- Verstichel, J., De Causmaecker, P., Spieksma, F., Vanden Berghe, G. (2012). The ship placement problem: Exact and heuristic approaches. The 2012 International Conference on Logistics and Maritime Systems. Bremen, Germany, 22-24 August 2012.

- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2012). The lock scheduling challenge: Improving efficiency through optimization and decision support. WCTRS - SIG2: KEY DEVELOPMENTS IN THE PORT AND MARITIME SECTOR, 21-22 May 2012.

- Verstichel, J., Vanden Berghe, G., Callens, H., Fredrick, F. (2010). A pooling approach for the feed mixing problem. Proceedings of ORBEL 24. ORBEL. Liege, 28-29 January 2010, 78-79.

- Wauters, T., Verstichel, J., Verbeeck, K., Vanden Berghe, G. (2010). A hybrid learning and combinatorial optimization approach for automotive maintenance scheduling. Proceedings of ORBEL 24. ORBEL. Liege, 28-29 January 2010, 107-108.
- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2010). A Best Fit Heuristic for the Lock Scheduling Problem. International Conference Operations Research Munich 2010: Program and Abstracts. International Conference on Operations Research. Munich, Germany, 1-3 September 2010, 229.
- Verstichel, J., Vanden Berghe, G. (2009). Late Acceptance Multiple Neighbourhood Search for Lock Planning. Proceedings of ORBEL'09. ORBEL. Leuven, 05-06 February 2009, 84.
- Wauters, T., Verstichel, J., Verbeeck, K., Vanden Berghe, G. (2009). A Learning Metaheuristic for the Multi Mode Resource Constrained Project Scheduling Problem. Learning and Intelligent OptimizatioN. Trento (Italy), 14-18 January 2009.
- Verstichel, J., Vanden Berghe, G. (2009). A Late Acceptance metaheuristic for the Lock Scheduling Problem. Book of Abstracts of the 14th Belgian-French-German Conference on Optimization (BFG'09). Belgian-French-German Conference on Optimization. Leuven, 14-18 September 2009, 174.

Meeting abstracts, presented at other scientific conferences and symposia, published or not published in proceedings or journals

- Wauters, T., Verstichel, J., Vanden Berghe, G. (2013). Two- and three-dimensional strip packing: a shaking procedure. Proceedings of ORBEL27. ORBEL. Kortrijk, 7-8 February 2013.
- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2013). The ship placement problem: Decision support through exact decomposition. ORBEL. Kortrijk, 7-8 February 2013.
- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2012). The lock scheduling problem: An exact approach. ORBEL26: Booklet of Abstracts. ORBEL. Brussels, 2-3 February 2012.

- Verstichel, J., De Causmaecker, P., Vanden Berghe, G. (2011). An improved heuristic for the orthogonal stock-cutting problem. Proceedings of the 25th ORBEL conference. ORBEL. Gent, Belgium, 10-11 February 2011, 151-152.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
CODES GROUP
Celestijnenlaan 200A box 2402
B-3001 Heverlee