Online lock scheduling and disruption management

R. Hageman

Online lock scheduling and disruption management

by

R. Hageman

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday October 11, 2021 at 9:00 AM.

Student number:	4483561	
Project duration:	February 1, 2021 – October 11, 2021	
Thesis committee:	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Dr T. van Essen,	TU Delft
	Prof F. Spieksma,	TU Eindhoven
	Dr T. Tutenel,	Macomi B.V.

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

TU Delft

Abstract

The Port of Antwerp is the second-largest container port in Europe. The rising demand for container transport requires significant investments in infrastructure projects. Macomi helps the Port of Antwerp to determine the effect of different projects on the throughput of the port using simulation. In this thesis, the aim is to create algorithms for the introduced online variant of the lock scheduling problem which are applicable for a real-time simulation. In addition, an algorithmic approach to recover the existing schedule when a vessel is delayed is required. To achieve these goals, three online lock scheduling algorithms are introduced and tested on realistic problem instances. Their run-time is negligible compared to exact methods and the resulting lock schedules are competitive. Assuming a constant lockage duration during scheduling allows the number of interactions to be reduced significantly with a small decrease in lock schedule quality. The online lock scheduling algorithms could also be applied to the problem of disruption management. The results are comparable to the high-performing adaptive large neighbourhood search meta-heuristic.

Contents

1 Introduction

Locks are used in different settings along the waterway. They allow vessels to traverse inclining landscapes and other obstacles like hydroelectric dams. By ensuring a constant water level in ports they smooth the process of loading and unloading. Inefficient scheduling of lock operations can add severe delays to the journey of a vessel when the demand is high. Delays of up to three days are reported at the Three Gorges Dam in China during peak season.

The Port of Antwerp is the second-largest container port in Europe and expects an ever-increasing demand for the foreseeable future. Infrastructure projects are costly and have a long lifespan. It is, therefore, crucial to identify possible bottlenecks early and resolve them efficiently. By simulating the arrivals of vessels and their movements throughout the port realistically, this can be achieved. In addition, simulation can aid in determining the effectiveness of said projects to resolve the bottleneck.

1.1. Problem statement

This thesis is a result of an internship at Macomi, a company that focuses on optimisation and simulation of complex problems. Together with the Port of Antwerp, they are creating a simulation of the complete port to create the aforementioned tool. The arrivals are simulated based on historical data and projections of future demands. Each vessel traverses the port according to realistic sailing rules. The problem left to solve is to create an efficient schedule for all of the locks and to resolve conflicts once a vessel is not able to arrive at the lock as scheduled due to a delay. An efficient lock schedule minimises the waiting time and the number of lockage operations. In addition, it ensures that there are no outliers with extreme delays.

In the simulation, vessels need to traverse several restricting elements, like tide windows, swing zones and locks. Because of the limited manoeuvrability of large sea-going vessels, they must have a plan to travel from one place to another without having to wait. Due to the relation with other schedules and the nature of arrivals, it is required to solve an unexplored version of the lock scheduling problem where vessels arrive over time and need to be added to the existing lock schedule. The first part of this work introduces algorithms for this online lock scheduling problem where the information of all arrivals is not known at the start.

In the second part of this thesis, disruption management is treated. Once a lock schedule has been constructed, vessels could arrive at other moments than planned. When this occurs, it is important to find a change in the schedule that accommodates this delay while also minimising the existing objective and the number of vessels affected by it.

1.2. Research questions

The main research objective of this work is two-fold. First, it is to create efficient algorithms for the online lock scheduling problem. Then, it aims to compare different solution methods on their ability to resolve disruptions. The main research question is as follow;

Which solution methods prove to be the most effective to automate the lock scheduling and real-time disruption recovery for the Port of Antwerp?

A set of sub-questions have been defined to help answer the main research question based on each part of the objectives. The following questions help to answer the lock scheduling part;

1. How can the lock scheduling problem posed by the Port of Antwerp be defined as an optimisation problem?
2. What algorithms can solve the defined lock scheduling problem?
3. How can problem abstraction during scheduling contribute to fewer interactions with vessels?

For the disruption management part, the following sub-questions are defined;

1. What algorithms can be used for disruption management?
2. Which parameters do affect the quality of the recoveries?

1.3. Contributions

In this thesis, a new variant of the generalised lock scheduling problem relevant for the Port of Antwerp is introduced. From the literature research, it is concluded that existing solution methods are not sufficient to solve the posed problem. A framework for online lock scheduling algorithms is introduced and three algorithms using this framework are presented. It is shown that the algorithms create competitive solutions with exact algorithms on small problem instances. By abstracting the lockage duration during scheduling, the number of interactions with vessels is limited. A systematic approach to convert an abstract lock schedule into a detailed lock schedule is presented. Experiments show that this reduces the interactions by approximately one third while increasing the average and maximum waiting time by 10% depending on the objective function used.

The introduced online lock scheduling algorithms are applied to the problem of disruption management and compared to an adaptive large neighbourhood search meta-heuristic. One of the algorithms performs similar to the meta-heuristic based on the number of disagreements on the placement of the disrupted vessel for random disruptions on realistic lock schedules.

1.4. Document structure

This section explains the structure of the remaining document. Chapter 2 starts with providing a more elaborated explanation of the lock scheduling problem and the variant examined in this work. Then, chapter 3 presents the existing methods found in the literature regarding lock scheduling, disruption management and related problems. In chapter 4 a data-analysis on all the lockages of 2019 in the Port of Antwerp is performed. Algorithms for the online lock scheduling problem are presented and analysed in chapter 5. Then, in chapter 6 the same algorithms are applied for disruption management and compared to a meta-heuristic. The results of the different experiments are summarised and discussed in chapter 7. Finally, chapter 8 concludes this thesis by answering the research questions and providing future research directions.

2

Problem statement

In this chapter, the lock scheduling problem is explained. First, a definition of the generalised lock scheduling problem is provided. Then, some common abstractions are explained. Finally, the lock scheduling situation at the Port of Antwerp and the lock scheduling problem that is the focus of this work is presented as a mathematical model and corresponding objective functions.

2.1. The generalised lock scheduling problem

The generalised lock scheduling problem consists of three components; locks, chambers and vessels. There are one or more consecutive locks, each with its own set of parallel chambers. Locks may have both homo- and heterogeneous chambers. A chamber is defined by its length, width and processing duration. Similarly, every vessel has a length, width and speed. Additionally, each vessel has to pass through one or more consecutive locks in either upstream or downstream direction. Figure 2.1 systematically shows the possible flows of a vessel. Vessels can enter and leave the waterway at any point between locks.

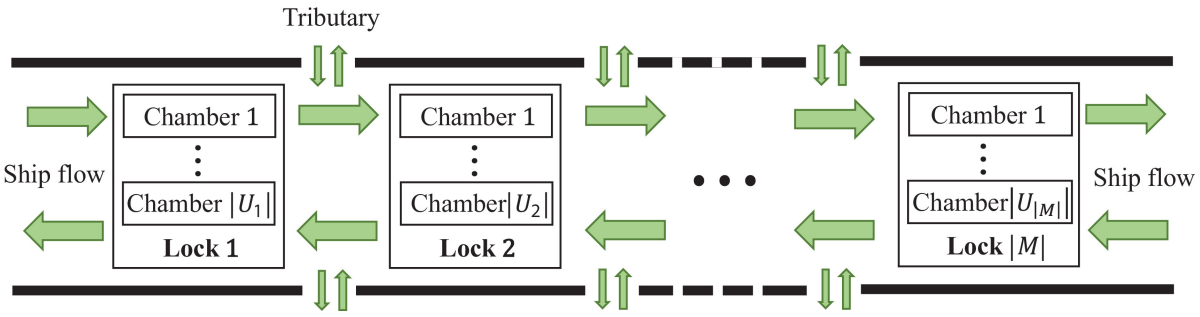


Figure 2.1: Schematic illustration of the generalised lock scheduling problem [1].

The placement of multiple vessels inside a chamber is called a lockage and computing the optimal lockage allocation is related to a well-known NP-complete problem, 2D bin-packing, with additional constraints. Although vessels are in reality not a true rectangle, they can be represented as such when safety margins are taken into account. Also, when placing vessels not adjacent to the quay it is required to secure it to a larger vessel. However, it is forbidden to secure onto a sea-going vessel [2]. Verstichel and Vanden Berghe [3] composed an extended overview of all the different requirements including illustrations.

Scheduling the sequence of lockages is less restricted. As vessels can overtake each other, the lockage at each lock can contain different sets of vessels. However, it is required that the vessel must be able to travel between the locks in the allocated period.

The purpose of a lock scheduling algorithm is to have vessels wait less at locks. Therefore the objective function can be defined by the (weighted) waiting time or tardiness of a vessel for a single

lock. When a vessel has to traverse several locks the time it is in the system from start to finish becomes relevant.

An objective with only the total waiting time will create schedules where some vessels are delayed severe to allow others to pass through fast. Sometimes a first-come-first-serve constraint is enforced to ensure fairness among the vessels. However, it is also possible to prevent large peaks in waiting time by including the maximum waiting time or squaring the individual delays.

Complementary to these objective components is water usage. There are several cases when this becomes a significant factor. For example in cases of drought or to prevent salinization at locations where a lock separates salty from freshwater. The water usage can be measured by the number of lockages.

The objective function is a proxy of the desired properties of an efficient lock schedule. The goal is to minimise the waiting time and number of lockages while also ensuring fairness among the vessels as outliers are undesirable. However, the weights provided to each of the components of the objective function can affect the absolute value. It is therefore not sufficient to compare algorithms solely on their relative objective function.

2.2. Lock scheduling abstractions

The lock scheduling problem gains its complexity mainly from two components; the vessel placement problem and the variable lockage duration based on the vessels allocated in the lockage. There exist abstractions of the lock scheduling problem which deal with these matters to reduce the complexity of the problem.

The vessel placement problem is solved to ensure that a lockage never exceeds the capacity of the chamber. Instead of defining the capacity of a chamber by its length and width, it is also possible to define the number of vessels it can fit. This is especially useful when the vessels are homogeneous in their sizes or when a few groups of vessels can be defined. Finally, some algorithms treat the capacity of a chamber as infinite.

The duration of a lockage operation could be set to a fixed duration to avoid changing it continuously when adding vessels. This is reasonable when the entering and exiting duration's are negligible compared to the processing duration. It also depends on the context and types of vessels. When vessels can enter the lock at the same time, this can be accurately estimated. However, large vessels, that require tugboats, take more time and are less manoeuvrable.

2.3. The lock scheduling problem at the Port of Antwerp

The Port of Antwerp is reachable from the North Sea through the river called the Westerschelde and welcomes all types of vessels, ranging from barges to large sea-going vessels. A large part of them needs to pass a lock. However, there is also traffic towards berths that are next to the river. The different locks at the Port of Antwerp are highlighted in Figure 2.2 and Table 2.1 shows their characteristics. The first four locks provide access to the right bank of the port. Under normal circumstances, the Van Cauweleartsluis is reserved for barges only. Because there is only data available about sea-going vessels, barges and this lock are omitted from this work.

Table 2.1: Characteristics of the locks at the Port of Antwerp.

Lock	Length (m)	Width (m)
Zandvlietsluis	500	57
Berendrechtsluis	500	68
Boudewijnsluis	360	45
Van Cauweleartsluis	270	35
Kieldrechtsluis	500	66
Kallosluis	360	50

Large sea-going vessels are not easily manoeuvrable and need to plan their journey along the river from the North Sea towards the port. Therefore, a lock schedule is created with a rolling horizon of at least 12 hours. To prevent large disruptions affecting the schedule, it is required to have 15 minutes of

buffer between every lockage during scheduling.

Throughout the day, new vessels are planning their journey through the port. It is not possible to stop already started processes, and completely altering the existing schedule will cause issues with the schedules of other parts of the port. It is therefore required to add new vessels to the existing schedule and only slightly affects the vessels planned around it. This type of scheduling where new vessels are added to an existing schedule is called online scheduling and is different from creating the optimal schedule provided with all the arrivals from the beginning.

The simulation currently envisioned by Macomi has online algorithms for all the different bottlenecks in the port. Examples of this are the berths and locks but also tide windows and narrow passages. A vessel becomes an agent which has to sail along the river and port according to the relevant schedules in real-time. Movements between two locks do occur at the Port of Antwerp. However, due to the simulation design and the bottlenecks between locks, this is handled by a backtracking path-finding algorithm which is not part of this thesis.

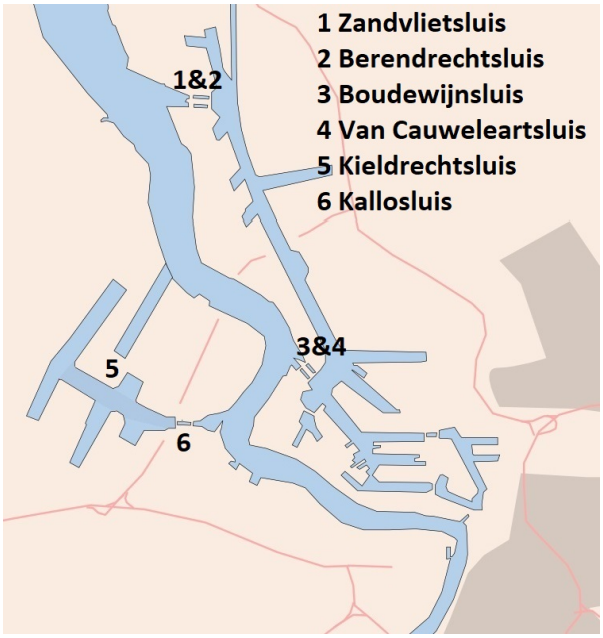


Figure 2.2: Map of the Port of Antwerp with each lock highlighted.

.

2.4. Formal problem definition

In the posed online lock scheduling problem there is a single lock with m heterogeneous chambers and a sequence of n vessels that arrive online. Let $C = \{c_1, ..., c_m\}$ be the set of chambers. Then, let L_c be the length, W_c the width and P_c the processing duration of chamber $c \in C$.

Let $V = \{v_1, ..., v_n\}$ be the set of vessels. With l_v the length, w_v the width and p_v the entry and exit duration of vessel $v \in V$. In addition, the arrival of vessel v is defined by its direction d_v and the arrival time a_v. There are two types of directions, either up- or downstream.

Let $L_c = \{l_1, ..., l_k\}$ be the set of lockages processed by chamber $c \in C$ and $L = \bigcup_{c=1}^{m} L_c$ be the set of all lockages. Then, let d_l be the direction of lockage $l \in L$ and s_l and e_l its start and end times.

Each chamber, c, processes its set of lockages, L_c, in alternating direction. Consecutive lockages l and l' must adhere to the following set of constraints where B is a constant buffer between sequential lockages;

$$s_{l'} \geq e_l + B \tag{2.1}$$

$$d_l \neq d_{l'} \tag{2.2}$$

Let $V_l = \{v_1, ..., v_n\}$ be the set of vessels processed by lockage l. A vessel can only be processed by a lockage if the lockage is in the same direction as the vessel arrives. Equations (2.3) and (2.4)

ensure that every vessel is processed by a single lockage. Equation (2.5) enforces that the direction of a vessel and the lockage it is placed in are in the same direction.

$$V_l \cap V_{l'} = \emptyset \qquad \forall l, l' \in L \tag{2.3}$$
$$v \in V_l \qquad \exists l \in L, \forall v \in V \tag{2.4}$$
$$d_v = d_l \qquad \forall v \in V_l, l \in L \tag{2.5}$$

The duration of a lockage is dependent on its chamber and the vessels it processed. In addition, when vessels arrive not exactly after each other, the duration increases. Each vessel retrieves a requested time of arrival (ra_v). Equation (2.6) ensures that the requested time of arrival is after the first moment the vessel actually can arrive. Equation (2.7) ensures enough time between the sequential arrival of the vessels v and v'. The start and end time of a lockage are defined by Equations (2.8) and (2.9). Finally, Equation (2.10) is an helper variable that is used in the objective function.

$$ra_v \geq a_v \qquad \forall v \in V \tag{2.6}$$
$$ra_{v'} \geq ra_v + p_v \tag{2.7}$$
$$s_l = min_{v \in V_l}(ra_v) \qquad \forall l \in L \tag{2.8}$$
$$e_l = max_{v \in V_l}(ra_v + p_v) + P_c + \sum_{v \in V_l} p_v \qquad \forall l \in L_c, c \in C \tag{2.9}$$
$$e_v = e_l \qquad \forall v \in V_l, l \in L \tag{2.10}$$

A schedule requires to have a fixed position for each vessel in a lockage. This position is defined by a x_v and y_v coordinate relative to the bottom left of the chamber. Equations (2.11) and (2.12) ensure that a vessel is placed within the dimensions of the chamber. A vessel must be moored onto the left or the right side of the chamber. The binary decision variable ml_v indicates if vessel v is moored onto the left side. Equation (2.13) enforces the correct x coordinate of vessel v based on the side it is moored on.

$$0 \leq x_v \leq W_c - w_v \qquad \forall v \in V_l, l \in L_c, c \in C \tag{2.11}$$
$$0 \leq y_v \leq L_c - l_v \qquad \forall v \in V_l, l \in L_c, c \in C \tag{2.12}$$
$$x_v = \begin{cases} 0 & \text{if } ml_v = 1, \\ W_c - w_v & \text{if } ml_v = 0 \end{cases} \qquad \forall v \in V_l, l \in L_c, c \in C \tag{2.13}$$

When multiple vessels are positioned within a lockage, they are not allowed to overlap. The functions $same\_side(v, l)$ returns the vessels in the same lockage l on the same side of vessel v. Between vessels $v, v' \in V_l$, it is required to have a minimum vertical distance of $vdistance(v, v')$ when moored onto the same side of the chamber. Equations (2.14) and (2.15) ensure that two vessels on the same side do not overlap and that the minimum distance is enforced.

$$|y_v - y_{v'} - l_{v'}| \geq vdistance(v, v') \qquad \forall v' \in same\_side(v, l), v \in V_l, l \in L \tag{2.14}$$
$$y_v \leq y_{v'} + l_{v'} \vee y_v \geq y_{v'} + l_{v'} \qquad \forall v' \in same\_side(v, l), v \in V_l, l \in L \tag{2.15}$$

When vessels share the same vertical position on opposite sides of the chamber, a minimum horizontal distance between them is required. The function $opposite\_side\_overlapping(v, l)$ returns the set of vessels in l that are moored on the opposite side of vessel v and share some overlapping y position. The horizontal distance required between these vessels are retrieved with $hdistance(v, v')$. Equation (2.16) enforces the additional horizontal distance for these vessels.

$$|x_v + l_v - x_{v'}| \geq hdistance(v, v') \qquad \forall v' \in opposite\_side\_overlapping(v, l), v \in V_l, l \in L \tag{2.16}$$

Equations (2.17) to (2.25) summarise the different decision variables and list their domain. The requested arrival times of a vessel at a lock is defined up to a minute. This can be modelled as an integer since the first event in the problem. It is therefore also required to provide the parameters as

an integer representing the minutes since the same moment.

$$d_v, v \in V, d_v \in \{0, 1\} \tag{2.17}$$
$$x_v, v \in V, x_v \in \mathbb{Z}, 0 \leq x_v \leq max_{c \in C}(W_c) \tag{2.18}$$
$$y_v, v \in V, y_v \in \mathbb{Z}, 0 \leq y_v \leq max_{c \in C}(L_c) \tag{2.19}$$
$$ra_v, v \in V, ra_v \in \mathbb{Z}, 0 \leq ra_v \tag{2.20}$$
$$e_v, v \in V, e_v \in \mathbb{Z}, 0 \leq e_v \tag{2.21}$$
$$V_l, l \in L, V_l \subseteq V \tag{2.22}$$
$$d_l, l \in L, d_l \in \{0, 1\} \tag{2.23}$$
$$s_l, l \in L, s_l \in \mathbb{Z}, 0 \leq s_l \tag{2.24}$$
$$e_l, l \in L, e_l \in \mathbb{Z}, 0 \leq e_l \tag{2.25}$$

2.5. Objective function

With the formal model definition presented, it is possible to define the different objective functions introduced in Section 2.1 more precise. The objective functions presented in this section are used for the different experiments throughout the thesis. The different objective function component weights (K_i) are defined at each experiment. Each objective function includes the number of lockages. They differ in their method of measuring delays and whether outliers are avoided by squaring the individual delays or by including the maximum delay.

1. **Squared waiting time:** $K_1 \sum_{v \in V}(ra_v - a_v)^2 + K_2 \sum_{l \in L}(1)$
2. **Squared tardiness:** $K_1 \sum_{v \in V}(e_v - a_v)^2 + K_2 \sum_{l \in L}(1)$
3. **Summed waiting time:** $K_1 \sum_{v \in V}(ra_v - a_v) + K_2 max_{v \in V}(ra_v - a_v) + K_3 \sum_{l \in L}(1)$
4. **Summed tardiness:** $K_1 \sum_{v \in V}(e_v - a_v) + K_2 max_{v \in V}(e_v - a_v) + K_3 \sum_{l \in L}(1)$

3

Literature review

In this chapter, the existing approaches related to the posed problem are evaluated. First, the literature related to the lock scheduling problem is presented. Subsequently, the disruption management part is assessed. In both sections, the differences between the posed problem and the problem solved by the existing methods are highlighted. Additionally, related scheduling problems are evaluated in both sections.

3.1. Lock scheduling

In this section, the literature on the generalised lock scheduling problem is presented. First, the algorithms that solve a version of the problem in polynomial time to optimality are examined. After providing an NP-hardness proof for a variant of the problem, the exponential exact algorithms are presented. Heuristics are also used to find solutions within a limited amount of time. Finally, solutions for related problems are assessed.

3.1.1. Polynomial time algorithms

The lock scheduling problem for a single chamber with constant lockage duration has been analysed by Hermans [4]. Its dynamic program finds the optimal solution in polynomial time when the capacity of the lock is limited to a single vessel. The same problem without capacity constraints is compared to the job shop scheduling problem by Passchyn et al. [5]. It concludes that the complexity of the problem cannot be derived from the available literature. It provides a polynomial-time algorithm based on the shortest path in an acyclic graph. Additionally, it proves that problems with an arbitrary chamber capacity represented by an integer can still be solved in polynomial time when enforcing a first-come-first-serve constraint.

The problem with constant lockage duration is extended to multiple chambers by Passchyn et al. [6]. A dynamic program to minimise the total waiting time in polynomial time is presented, given that the number of chambers is a constant. Again it is assumed that the capacity of a lock can be represented in terms of a number of vessels.

3.1.2. Exponential algorithms

The previous section presented some of the polynomial-time algorithms for the lock scheduling problem with abstractions. However, there are NP-hardness proofs for slightly more complicated versions of the problem. Passchyn and Spieksma showed that minimising the total waiting time for two identical sequential locks with a constant lockage duration is NP-hard even if every vessel travels in the same direction [7]. The existence of exponential algorithms for the more general variants of the problem comes therefore at no surprise.

Verstichel et al. present a mixed-integer linear program (MILP) for the lock scheduling problem with a single lock with heterogeneous chambers [8]. This work is extended by the same authors in terms of a benders decomposition [9]. It reduces the run-time by adding vessels to lockages and lockages to chambers using a MILP and solving the vessel placement sub-problem separately. Additionally, it

further reduces the run-time by replacing the exact vessel placement algorithm with a heuristic called multi-order best-fit.

Ji et al. extended the formulation of Verstichel et al. to include sequential lock movements [10]. The same authors additionally formulated another model for the same problem based on a multi-commodity network [11]. Experiments using these formulations show an unpredictable run-time which can reach over 16 hours for problems with 20 vessels and two chambers.

3.1.3. Heuristics

All exact exponential algorithms in the previous subsection were formulated as a MILP. Van Adrichem tried to improve such formulations by splitting the search space into small chunks and solving them separately [12]. It is not able to outperform the baseline during busy moments.

Ji et al. noticed similarities between the lock scheduling problem and the well-studied vehicle routing problem [13]. A large neighbourhood search, well known within the vehicle routing problem literature, is used to solve the problem for a single lock. Destroy and repair operations are based on general operations used for other problems.

Neighbourhood search techniques have also been applied to the lock scheduling problem by others. However, these approaches make more simplifying assumptions than Ji et al. Prandtstetter et al. solves the interdependent lock scheduling problem with identical chambers [14]. It presents operations that destroy and repair a schedule in one pass and are specifically crafted for the problem at hand.

Verstichel et al. presented a neighbourhood search for the lock scheduling problem with a single lock with identical chambers where the solution is defined by an ordering of vessels [15]. Provided with this ordering, heuristics will determine the feasibility of the vessel placement sub-problem and the start time is defined by the latest arriving vessel. Verstichel and Vanden Berghe explored the same problem and approach but applied late acceptance criteria [16].

When solving the generalised lock scheduling problem it is required to determine a position for every vessel inside the lock. Verstichel et al. introduced different methods for this [2]. First, it presents an exact MILP which also formed the basis of several of the exponential algorithms presented earlier. Then two heuristics are presented which are not computationally intensive and produce solutions with a small optimality gap. One of these heuristics is used in combination with the aforementioned benders decomposition of an exact MILP.

3.1.4. Related problems

Zhang et al. examined another online problem of scheduling the passage of vessels on a restricted 2-way waterway where large vessels cause other vessels from the opposite direction to wait for them [17]. It investigated the influence of different priority rules and concluded that significant improvements can be made. However, the difference between these waterways and locks are evident. At a lock, an alternative behaviour is required between upstream and downstream while the improvement on the waterways is gained by grouping vessels in the same direction.

The lock scheduling problem can be represented as a flexible job shop scheduling problem with parallel batch processing machines, incompatible job families and job dependent set-up times combined with the vessel placement sub-problem. MILP and constraint programming are common approaches to solve such problems [18]. However, because of their computational complexity and unpredictable run-time, they are no good fit for online problems. Therefore, often, dispatching rules are developed which act as a static online heuristic. These rules can be created by a human with experience but could also be learned automatically. Jun et al. creates training and validation sets using exact methods and trains a random forest to create such dispatching rules [19].

In terms of traffic in both directions, it is possible to compare a landing strip with a locks chamber. In this sense, an airport can be compared with a lock of multiple chambers which are individually operated. There are also differences. A locks chamber has an alternating direction of operations and can process multiple vessels at a time. Continuous operation in the same direction is possible at a strip and sometimes even required [20]. However, the main difference is that planes can wait before entering the strip. This can either be at the terminal or in the sky.

3.1.5. Conclusion

From the literature on the lock scheduling problem, it is clear that the problem at hand is hard to solve and no exact polynomial algorithm is to exist for the offline and online case. Additionally, except for the

related job shop scheduling problem, no algorithms for online lock scheduling exist.

3.2. Disruption management

The lock scheduling problem only recently gained traction compared to other parts of maritime and port related optimisation problems. To the best of my knowledge, no definition of a disruption management problem or solution approach exists in the literature. Therefore, only the approaches for disruption management on related problems are examined in this section.

3.2.1. Related problems

Airline operations can be disrupted due to several factors. Resources of the airline, for example, crew, aircraft or fuel, could become unavailable. In addition, external factors could affect the operations. Examples of this are the weather and traffic. Su et al. created a survey of the possible causes, mitigations and methods for disruption management for airlines [21]. Methods are divided between solving either disruptions for solely aircraft, crew or both. Often exact approaches are used for the individual disruptions. However, neighbourhood search techniques are also applied for integrated disruption management. The objective function for these decisions is clearly defined when the costs of each decision are known. However, a majority of the mitigations do not apply to vessel disruption management. It is not possible to cancel a vessel that is already on the river towards the port. In addition, such vessels are less manoeuvrable in comparison to aircraft that can wait in both the air and at an airport. Also, cargo is cannot be transferred to other vessels.

Subramaniam et al. [22] survey disruption management in the context of job-shop scheduling. The survey lists 17 different commonly studied disruptions. From this list, five general disruptions with their repair operation are identified. After applying the repair operation to the affected job, sequential jobs can be right-shifted until no jobs are overlapping anymore [23]. Subramaniam et al. present a heuristic where the disrupted job gets rescheduled to its new optimal position [24]. While the lock scheduling problem is related to the job shop scheduling problem, these approaches do not incorporate batching of jobs which is similar to placing vessels together in a lockage.

3.2.2. Conclusion

Disruption management is unexplored for the lock scheduling problem in the literature. However, it is well studied for other types of problems. The significant differences between the types of mitigations and the freedom to apply them, rule them unusable for lock scheduling.

4

Data analysis and problem instance generation

In this chapter, the data provided by the Port of Antwerp about all lockages of 2019 is analysed. Then it is explained how this data is used to generate realistic problem instances which are used in experiments in the next chapters.

4.1. Data analysis

The Port of Antwerp provided a data-set containing all the lockages with the sea-going vessels of 2019. It reports the moment the doors of the lock opened and closed and the arrival and departure of every vessel. In addition, the maximum length and width of each vessel are known. As there is no data available about barges, the Van Cauwelaertsluis is not included in this analysis.

Figure 4.1 presents details about the lockages and the number of vessels transferred, split out over different time windows and by each lock. The first two graphs show that the number of lockages processed per lock per day is roughly similar for every lock. However, the Kieldrecht- and Kallosluis transfer fewer vessels per day. From the third graph, it is clear that the majority of the lockages contain only a single vessel. Although, it could occasionally reach up to seven.

The final three graphs present the trends of vessels processed over different periods. During the fourth month of 2019, the Kieldrechtsluis was closed due to maintenance. This explains both the sharp decrease and increase for the Kieldrechtsluis and Kallosluis respectively. The latter processed the vessels of the former as they both lead to the left bank. Besides a small reduction in vessels in the morning, there seem to be no other seasonal effects.

4.2. Realistic problem instance generation

This section explains the process of generating problem instances that will be used in the subsequent chapters. First, some general remarks regarding the problem instances are made. Then the method of converting the historical arrivals directly into a problem instance is treated. Finally, the patterns in the historical data are used together with demand projections to generate realistic arrivals.

The Port of Antwerp has two banks that are reachable by different sets of locks. Each bank has a main point of entry where the largest locks are situated. For the left bank, this is the Kieldrechtsluis. For the right bank, they are the Berendrechtsluis and Zandvlietsluis. The algorithms designed in this work are tailored to create lock schedules for a single lock with multiple chambers. Therefore, each bank is reduced to its main entry point during experiments. By redirecting the traffic of the other locks towards these main locks, the demand will become larger than in reality. This is by no means a simplification of the problem. In addition, it resolves the problem with the closure of the Kieldrechtsluis as all the traffic was redirected to the Kallosluis. These vessels are now redirected back to the Kieldrechtsluis.

Provided with this set of vessels that need to be processed by a lock, it is important to determine a realistic moment for which the vessel requests to be scheduled in the lock. When using the realised arrival times as input for the lock scheduling algorithms it becomes easy to generate a schedule. To

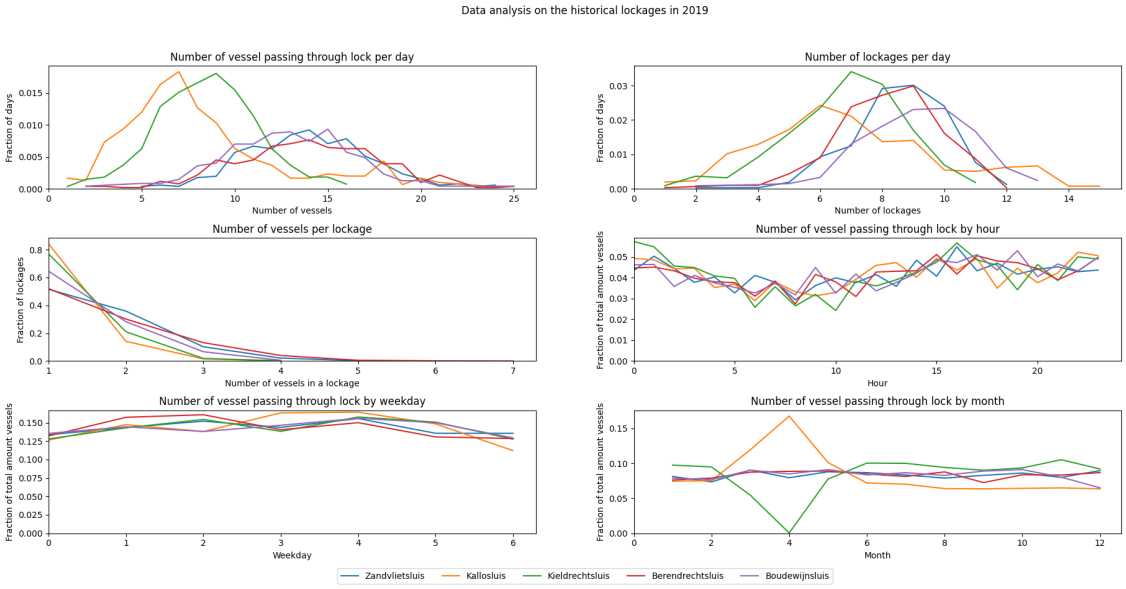


Figure 4.1: Data analysis performed on the data-set provided by the Port of Antwerp containing all historical lockages of 2019.

prevent this, the times a vessel is arriving at the start of the Westerschelde is used instead. This will spread out the vessels as they do not travel the river at an equal speed and requires the algorithms to create a schedule by themselves.

Based on confidential yearly demand projections for the year 2030 and the hourly, daily and monthly patterns in the historical data of 2019, it is possible to generate additional arrivals. These arrivals are converted into a problem instance in the same manner as explained for the historical arrivals. In addition to the same patterns and the historical arrivals, it is possible to generate more problem instances similar to historical 2019.

Each experiment will explain how many problem instances are generated and the duration of these instances. Some experiments are faster than others and can create schedules for extended durations of a few months up to a year. For others, this might not be feasible or useful. In addition to the number of problem instances, each experiment states how the arrivals are ordered. It is possible to add vessels in chronological ordering. However, sometimes this does not hold in practice and a random ordering could be required.

5

Online lock scheduling

This chapter treats the online lock scheduling part of the research questions. First, three different online lock scheduling algorithms are presented that use an abstraction of constant lockage duration to reduce the number of interactions. Then, the applicability of a heuristic for the vessel placement sub-problem is examined. Using this heuristic the run-time of the algorithms is analysed theoretically and empirically on artificial instances. The algorithms will be compared with an exact offline algorithm on small problem instances and relatively on realistic size problems. Finally, the trade-off between solution quality and interactions with vessels is made by comparing the algorithms with an online algorithm that directly creates detailed lock schedules.

5.1. Online lock scheduling algorithms

This section introduces three different online algorithms for the posed lock scheduling problem. It does so, by first reiterating the different design constraints and explaining a framework in which the algorithms have to operate. Then, the different algorithms are presented.

5.1.1. Lock scheduling framework

Information about a vessel comes available over time when a vessel arrives close to the Port. It is therefore important to be able to decide on a single vessel. Algorithm 1 presents pseudo-code representing the framework in which all the algorithms will operate. During the first step, the lock scheduling algorithms return a set of proposals to accommodate the vessel. Then, out of all the options, a single proposal is selected. This is based on the objective function and the feasibility of the proposal regarding other schedules at the port. Finally, the selected proposal has to be executed onto the lock schedule to finalise it.

Algorithm 1 Framework for the online lock scheduling algorithms

Require: $vessel, arrivalTime, direction$
$allProposals \leftarrow$ DetermineAllProposals($vessel, arrivalTime, direction$)
$selectedProposal \leftarrow$SelectProposal($allProposals$)
ExecuteProposal($selectedProposal$)

Every proposal has to be evaluated before a selection can be made. It is trivial to determine the objective difference regarding the lock schedule. However, when the algorithms are used in a simulation of the complete port, this becomes more complex. It is therefore desirable to minimise the number of interactions with vessels. This is achieved by assuming a constant lockage duration during the scheduling phase. This allows the algorithms to add vessels to a lockage without the requirement of interacting with each vessel in that lockage to ensure the new duration is fine by them. At the end of this chapter, an experiment will be performed to analyse if this indeed reduces the number of interactions with vessels. But during the other experiments, a constant lockage duration of 60 minutes is assumed. This allows at least one vessel to be added to the lockage, have the lock move to the other direction

and have some additional time left. The number of vessels in the lockage may require more time than the constant lockage duration. During the conversion, sequential lockages are right-shifted to make room for such lockages.

5.1.2. Lock scheduling algorithms

Two online scheduling algorithms are created which differ in their abilities to affect other vessels and lockages than the vessel currently scheduled. Algorithm 2 presents the logic which each of these algorithms shares. When a vessel cannot be allocated with the current arrival time, the algorithm will delay the vessel and try again until at least a single solution is found. For a lockage to accommodate a vessel, it must traverse in the right direction, the vessel must arrive before the start of a lockage and the vessel placement sub-problem must be solved for the vessels already in the lockage together with the currently scheduled vessel.

The difference between the two online scheduling algorithms are implemented in the feasibility checks and explained as follows;

1. **Default:** This algorithm is not able to delay other lockages. A new lockage is therefore only created if there is enough time between the predecessor and the successor. As the lockage duration is assumed to be constant a vessel can always be added to a lockage if the vessel placement sub-problem can be solved.

2. **Non-Greedy:** This algorithm extends the default algorithm and is allowed to delay other lockages. Therefore it will always propose solutions for creating a lockage as it will push its successors onto later in time.

Algorithm 2 Basic algorithm for the online lock scheduling algorithms.

Require: $lockSchedule$, $vessel$, $arrivalTime$, $direction$
  if $lockSchedule$ is empty **then**
    Propose to create lockage at $arrivalTime$
    Return
  end if

  $lockageBeforeVesselArrival$ ←DetermineLockageBeforeVesselArrival($arrivalTime$)
  $lockageAfterVesselArrival$ ←DetermineLockageAfterVesselArrival($arrivalTime$)

  if DoesLockageAccomodateVessel($lockageBeforeVesselArrival$, $vessel$) **then**
    Propose to add vessel to $lockageBeforeVesselArrival$
  end if

  if DoesLockageAccomodateVessel($lockageAfterVesselArrival$, $vessel$) **then**
    Propose to add vessel to $lockageAfterVesselArrival$
  end if

  if CanCreateLockageAfter($lockageBeforeVesselArrival$, $vessel$) **then**
    Propose to create lockage after $lockageBeforeVesselArrival$
  end if

  if CanCreateLockageAfter($lockageAfterVesselArrival$, $vessel$) **then**
    Propose to create lockage after $lockageAfterVesselArrival$
  end if

  if no proposal is created **then**
    Delay vessel until end of $lockageAfterVesselArrival$ and reschedule it
  end if

5.1.3. Local improvement

Online algorithms are known to make sub-optimal decisions due to incomplete knowledge about the problem at the moment of decision making. To improve the quality of the resulting lock schedules, a third algorithm is introduced. This algorithm is called **Improved** and uses the same logic as the Non-Greedy algorithm. However, each proposal will be evaluated and if possible improved.

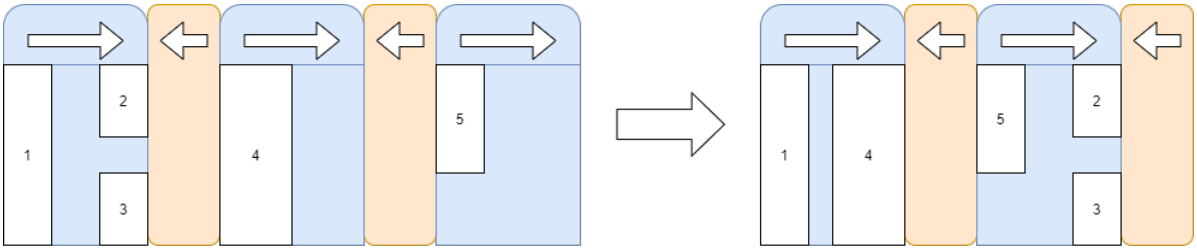


Figure 5.1: Example where the ordering in which vessels are added affect the number of lockages required to allocate all vessels.

Figure 5.1 shows that the ordering in which vessels are added to the lock schedule is important. If the vessels are scheduled in another order there would only be two lockages required. The first improvement aims to solve this problem. Whenever a new lockage is created for a single vessel, it tries to find different subsets of the neighbours of that lockage that fit together.

Another method to correct sub-optimal decisions based on the ordering of arrivals is changing two vessels based on their waiting time. When creating a new lockage for a vessel, it might be beneficial to replace that vessel with another vessel in an earlier lockage. This will not be effective when the arrivals are in chronological ordering. However, it allows revising the ordering in the schedule when the arrivals are more randomly ordered.

5.2. The vessel placement sub-problem

Each of the online lock scheduling algorithms often has to solve the vessel placement sub-problem. Due to the large variety of vessels arriving at the Port of Antwerp, it is not possible to represent the capacity of the locks with an integer like some algorithms in the literature. The Multi-Order Best-Fit (MO-BF) heuristic is a state-of-the-art heuristic to solve this NP-hard problem [2]. It extends the commonly used best-fit heuristics and derivatives by using different orderings in which the vessels are considered. Examples of such orderings are based on width, length and area.

To assess the applicability of MO-BF for the lock scheduling situation at the Port of Antwerp, it is compared with an exact algorithm that tries every possible permutation of the vessels. This is the algorithm currently used by Macomi. The algorithms are compared on their ability to recreate historical lockage and their run-time complexity.

5.2.1. Ability to recreate historical lockages

The algorithm used to solve the vessel placement sub-problem must create realistic results. This is validated by reconstructing historical lockages. It is expected that a significant part of the lockages can be reconstructed. The historical dataset with all the lockages of 2019 is used for this. It contains 14139 lockage of which 9126 transport only a single vessel. As these are trivially solvable given that the dimensions are correct, they are omitted from the validation.

Table 5.1: Overview of the required safety buffer during the vessel placement problem based on the length of a vessel.

Vessel length (m)	Safety buffer (m)
<80	5
<180	15
<250	20
≥250	30

To ensure safe operation of a lock, it is required to maintain buffers between vessels. Table 5.1 contains an overview of the safety buffers required in the vertical direction based on the length of

a vessel. Additionally, it is important to have at least 13 meters in the horizontal direction to allow tugboats to leave the lock. 39 lockages have been removed from the data-set as they contain one or more vessels which width does not fit inside the lock with the buffer for the tugboats. In reality, these lockages might have locked the tugboats along or managed to leave with a smaller safety distance.

Table 5.2 shows that both algorithms can recreate a significant proportion of the historical lockages. The heuristic fails on 5 additional cases. Each of these cases were lockages with three or more vessels. It is interesting to note that the exact algorithm is also not able to reconstruct every lockage.

Table 5.2: Results of the vessel placement sub-problem validation on historical lockages with two or more vessels in 2019.

Algorithm	Solved	Failed
Multi-Order Best-Fit	4810 (96,7%)	164 (3.3%)
Exact	4815 (96,8%)	159 (3.2%)

The experiment has been repeated for instances with at least three vessels and a gradually reduced safety buffer to mimic situations where the constraint is not enforced thoroughly. The results are presented in Table 5.3. It shows that the algorithms can reconstruct more lockages when the buffers are reduced. However, there remain a few lockages which both algorithms cannot solve even when all buffers are removed. As there are only a few of such cases, they are treated as input errors with the dataset.

Table 5.3: Results of the vessel placement sub-problem validation on historical lockages with three or more vessels in 2019 with reduced safety buffers.

Safety buffer reduction (m)	Failures of exact algorithm	Failures of multi-order best-fit heuristic
0	79 (6.38%)	84 (6.78%)
1	49 (3.95%)	51 (4.12%)
2	34 (2.74%)	35 (2.82%)
5	13 (1.05%)	13 (1.05%)
10	4 (0.32%)	4 (0.32%)
15	4 (0.32%)	4 (0.32%)
20	3 (0.24%)	3 (0.24%)
25	2 (0.16%)	2 (0.16%)
30	2 (0.16%)	2 (0.16%)

5.2.2. Run-time analysis

The run-time of both algorithms is analysed as follows. First, the time required to solve all the historical lockages is compared. Then a carefully constructed instance that can be indefinitely scaled is solved. For these experiments, the implementation of Imahori and Yagiura [25] is used. It has a theoretical run-time complexity of $\mathcal{O}(n \log n)$ due to the usage of a heap and a doubly-linked list to represent the skyline. All experiments are run on Windows 10 with an Intel i7 processor and 12GB of RAM and are implemented in C#.

Figure 5.2 shows the run-time of both algorithms for solving all historical lockages ten times. The ordering in which the vessels were provided to the algorithms is randomly shuffled. The results are presented per lockage size. It is clear that the algorithms are competitive for problems with 4 or fewer vessels. For larger lockages, the heuristic can find a solution faster.

In addition to the historical lockages, the algorithms are also compared on a problem that required the MO-BF heuristic to perform all its possible actions. It consists of three vessels of 3x2 and 2 vessels of 1x3. No buffers are assumed. As depicted in Figure 5.3, provided with a lock of 5x6, the algorithm completely fills the lock with vessels (grey) and wasted space (orange). It is therefore possible to increase the problem size by stacking multiple problems onto each other. This is achieved by increasing the length of the lock and adding more vessels with the same ratios. This is relevant for the case when the demand on the locks increases or when the barges are added to the simulation.

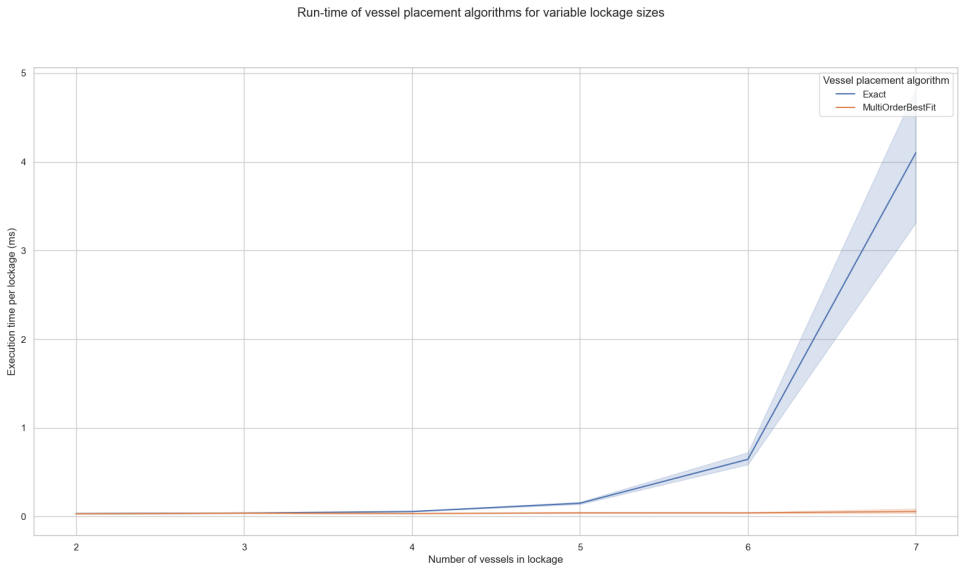


Figure 5.2: Run-time of the different algorithms for the vessel placement sub-problem on historical lockages.

The MO-BF heuristic can solve problems with up to 140 vessels within a millisecond. However, the exact algorithm requires between 4 and 5 seconds for problems with 10 vessels.

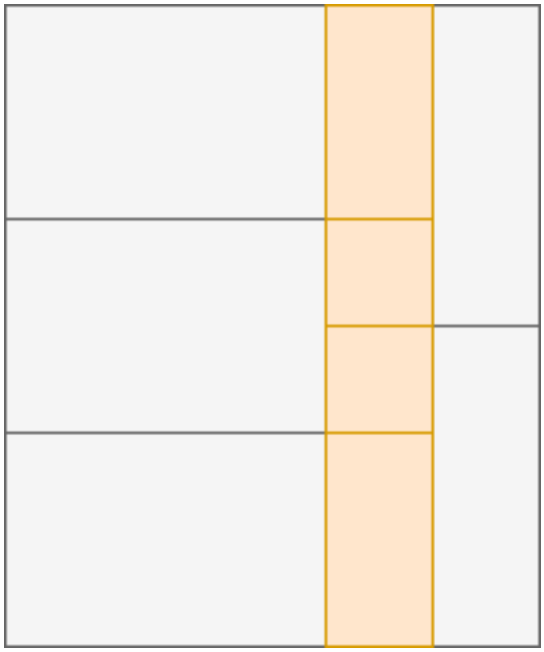


Figure 5.3: Example of the smallest problem instance of the empirical run time analysis for the vessel placement sub-problem.

5.3. Theoretical run-time analysis

Provided with the state-of-the-art heuristic to solve the vessel placement sub-problem it is possible to determine the theoretical run-time analysis of the algorithms. This starts with the basic algorithm and more details, including the local improvements, are added throughout this section. Then the run-time of the evaluation of the proposals and the execution of the selected proposal is analysed.

5.3.1. Creating proposals

The input of this algorithm consists of n vessels. In the worst case, each vessel is placed in its lockage causing n lockages. The pointers to each lockage in a linked list per chamber are stored in a self-balancing tree based on their start time. This allows the lockage after the arrival time to be determined with a run-time complexity of $\mathcal{O}(\log n)$. The neighbours of this lockage can be retrieved in constant time.

The multi-order best-fit heuristic has a run-time of $\mathcal{O}(k \log k)$ with k being the number of vessels in a lockage. In theory, every vessel could fit inside a lockage. However, as this is in practice lower we use k as the maximum number of vessels that can fit inside a lock.

Creating a new lockage only requires validating that enough time is between the predecessor and successor. Both lockages can be retrieved in constant time after the retrieval of the lockage after the vessel arrival due to the linked list.

The run-time complexity of the algorithm without recursion is $\mathcal{O}(logn + klogk)$. In the worst-case, each vessel arrives at the same time and in the same direction causing the algorithm to execute $2n$ recursive calls. This results in $\mathcal{O}(nlogn + nklogk)$. Finally, in the cases where a lock schedule for an extended period is created, we can assume that $k << n$ and the run-time complexity for the basic algorithm is reduced to $\mathcal{O}(nlogn)$.

The aforementioned run-time complexity holds for the Default algorithm. However, the Non-Greedy algorithm can delay an arbitrary number of lockages. Delaying a lockage is similar to determining the possibility of adding a new lockage in terms of checks. It can thus be performed in constant time. The run-time complexity of the algorithm without recursion is therefore changed into $\mathcal{O}(n + logn + klogk)$. Using the same number of recursive calls and the assumption about k results in a run-time complexity for the Non-Greedy algorithm of $\mathcal{O}(n^2)$.

5.3.2. Local improvement

The previous subsection provided a run-time complexity analysis for the Default and Non-Greedy algorithms. The Improved scheduler requires a different analysis because of its method of adapting existing proposals. For the following analysis, it is assumed that the number of returned proposals by the Non-Greedy algorithm can be limited by the constant G.

Algorithm 3 provides a detailed description of the first local improvement which combines three lockages into two. The predecessor of the new lockage is retrieved in $\mathcal{O}(logn)$ from the self-balancing tree. Provided with this lockage it is possible to retrieve the next two lockages in the same direction in constant time. This is due to the linked list and the fact that a neighbouring lockage in the same direction is at a maximum distance of two lockages.

Every vessel in the first predecessor that could arrive before the start of the second predecessor can be selected in $\mathcal{O}(k)$ due to a hash-map with the earliest arrivals of each vessel. In the worst case, every vessel in the predecessor and the arriving vessel could arrive before the predecessor of the predecessor. This results in $\mathcal{O}(2k + 1) = \mathcal{O}(k)$ vessels of which $\mathcal{O}(2^k)$ subsets can be created. For each subset, it is determined if the multi-order best-fit heuristic can place all the arrivals in the chamber which takes $\mathcal{O}(klogk)$. The same applies to the complement of the subset. Therefore the total run-time complexity of this local improvement is $\mathcal{O}(G2^k klogk + Glogn)$.

The second local improvement requires less effort to analyse and is provided in pseudo-code in Algorithm 4. Again the predecessor in the same direction of the new lockage gets retrieved in $\mathcal{O}(logn)$. Then, for every vessel inside the predecessor, it is determined if the arriving vessel can be replaced with it. All this requires is performing the multi-order best-fit heuristic for $\mathcal{O}(k)$ times. Therefore the total run-time complexity of this local improvement is $\mathcal{O}(Gk^2logk + Glogn)$.

The local improvements are executed once for every arriving vessel. Therefore the run-time complexity of the Improved lock scheduling algorithm is $\mathcal{O}(n^2 + nG2^k klogk + nGlogn + nGk^2logk + nGlogn) = \mathcal{O}(n^2 + nG2^k klogk)$. To conclude, a summary of all the run-time complexity results to create a schedule with n vessels is presented in Table 5.4.

5.3.3. Evaluation of proposals

During the simulation, selecting which action to execute depends on more than an objective function evaluation. The feasibility and objective function differences for the other schedules in the port need to be evaluated. A backtracking algorithm is used to find a feasible path for every vessel across the

Algorithm 3 Pseudo-code of local improvement which reduces three lockages into two.

Require: $lockSchedule$, $vessel$, $arrivalTime$, $direction$
  $predecessor \leftarrow$DetermineLockageBeforeVesselArrival($arrivalTime$)
  $firstInDirection \leftarrow$DeterminePredecessorInDirection($predecessor$, $direction$)
  $secondInDirection \leftarrow$DeterminePredecessorInDirection($firstInDirection$, $direction$)

  $vessels \leftarrow secondInDirection.Vessels \cup firstInDirection.vessels \cup vessel$
  $possibleVessels \leftarrow$DetermineAllVesselsWhichCanArriveAt($vessels$, $secondInDirection$)
  for each $subset \subset possibleVessels$ **do**
    if not MultiOrderBestFit($subset$) **then**
      Continue
    else if not MultiOrderBestFit($vessels \setminus subset$) **then**
      Continue
    else
      Propose to create lockage with subset and another with its complement
    end if
  end for

Algorithm 4 Pseudo-code of local improvement which swaps two vessels to reduce the tardiness.

Require: $lockSchedule$, $arrivingVessel$, $arrivalTime$, $direction$
  $predecessor \leftarrow$DetermineLockageBeforeVesselArrival($arrivalTime$)
  $predecessorInCorrectDirection \leftarrow$DeterminePredecessorInDirection($predecessor$, $direction$)
  for each $vessel \in predecessorInCorrectDirection$ **do**
    $vessels \leftarrow predecessorInCorrectDirection \cup arrivingVessel \setminus vessel$
    if MultiOrderBestFit($vessels$) **then**
      Propose to swap $vessel$ with $arrivingVessel$
    end if
  end for

different restricting elements. As this pathfinding algorithm will dominate the run-time of the evaluation, the run-time complexity analysis of the lock schedule proposal evaluation is omitted.

5.3.4. Execution of proposal

A lock schedule consists of various data structures to allow the efficient execution of operations. This is a trade-off between memory and run-time complexity. As stated earlier a self-balancing tree is used to store pointers to lockages in a linked list based on their start time. Additionally, there are hash maps used to point to the same pointers based on the lockage and vessel identifiers.

A proposal consists of different actions which have to be performed on the lock schedule. During the following analysis of these actions, it is assumed that the key-value pairs of the hash-maps are well distributed. This allows the assumption of constant time complexity for every operation on them.

1. **Create new lockage, l, at time t:** ($\mathcal{O}(logn)$) First, the lockage before time t has to be retrieved from the self-balancing tree which takes $\mathcal{O}(logn)$ time. The new lockage is in constant time added to the linked list. Finally, the linked list element is added to the self-balancing tree again in $\mathcal{O}(logn)$ time.

2. **Add vessel, v, to lockage l:** ($\mathcal{O}(1)$) The usage of the hash-map allows to retrieve the lockage in constant time. Adding the vessel to the lockage object and the hash-map connecting vessels with lockages also takes constant time.

3. **Remove vessel, v, from current lockage:** ($\mathcal{O}(1)$) The lockage of the vessel is retrieved in constant time from the hash-map. Removing the vessel to the lockage object and the hash-map connecting vessels with lockages also takes constant time.

4. **Update the duration or start time of lockage l:** ($\mathcal{O}(1)$) The mentioned attributes are values stored in the lockage which can be retrieved in constant time. The feasibility of such changes can

Table 5.4: Summary of the run-time complexity analysis results to create a schedule with n vessels by the online lock scheduling algorithms.

Algorithm	Run-time complexity
Default	$\mathcal{O}(nlogn)$
Non-Greedy	$\mathcal{O}(n^2)$
Improved	$\mathcal{O}(n^2 + nG2^k klogk)$

be determined in constant time by comparing the times of the neighbours.

As a lockage never gets removed, every proposal creates a single new lockage for the arriving vessel. This happens for a maximum op n times. Therefore the time complexity of the execution of the proposals throughout the complete simulation is $\mathcal{O}(nlogn)$.

5.4. Empirical run-time analysis

In this section, the theoretical run-time analysis of the previous section is validated. When analysing the run-time there are two instance characteristics of interest; the inter-arrival time controls how close the vessel arrive at each other and the plannings horizon determines over which period vessels are arriving. Both control the number of vessels and thus the amount of work the algorithms are required to perform. During the following experiments, the influence of these parameters on the run-time of the online lock scheduling algorithms is analysed.

To gain precise control over the relevant parameters, arrivals are generated for the following experiments. Arrivals will be sampled from a Poisson process. The dimensions of a vessel are uniformly random selected from any vessel in the 2019 data-set. The values for the parameters during the experiments are presented in Table 5.5 and are based on the data analysis on historical lockages. Arrivals are generated for both a single and double chamber lock. The dimensions are similar to those of the main entry points of the left and right bank respectively.

With these parameters, 25 instances will be generated for each parameter combination. Because each instance is a realisation of a random process the number of vessels per instance is not fixed. Therefore the run-time divided by the number of vessels is reported. The algorithms will solve every instance in both a chronological ordering and with 10 random arrival orders. All experiments are run on windows 10 with an Intel i7 processor and 12GB of RAM and are implemented in C#.

The framework for the online lock scheduling algorithms requires an objective function to select the best proposal. During these experiments, this objective is based on the summed waiting time objective function. Exact weights for each of these components are irrelevant for the run-time. No significant differences have been observed when the experiment is repeated with different objective functions and weights.

Table 5.5: Parameters with their default value and range for the run-time analysis of the online lock scheduling algorithms.

Parameter	Unit	Default value	Range
Inter arrival time	hours	0.6	[0.3, 0,35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
Horizon	days	2	[1, 2, 4, 8, 16, 32, 64, 128, 256]

5.4.1. Hypothesis

It is expected that increasing the horizon will cause a polynomial growth in the run-time for every algorithm. This is caused by maintaining the different data structures of the lock schedule. Increasing the horizon will cause these data structures to grow.

When the inter-arrival time decreases, vessels arrive closer to each other. This increases the number of possibilities to combine vessels inside a lockage. All algorithms are expected to have an increase in run-time due to the additional vessels. However, the additional vessel combination is expected to have a more severe impact on the Non-Greedy and especially the Improved algorithm. This is because these algorithms have the possibilities to move already placed vessels.

The parameters of the problems are equal for both locks. It is expected that instances with the single chamber lock require more time to be solved. Provided with an equal amount of vessels, a single chamber becomes more crowded as there is less room to spread the arriving vessels. This causes the algorithms to combine vessels more often together. Similarly, it is expected that a random ordering takes more time compared to the chronological ordering as there are more lockages to correct.

5.4.2. Results

Figures 5.4 and 5.5 show the run-time of the algorithms on the generated instances on the instances with both a single and a double chamber lock. The trends in the results are similar for both graphs.

Decreasing the inter-arrival time increases the run-time of the algorithms. This is especially noticeable for the improved algorithm. Increasing the horizon of the experiment increases the run-time linearly for all algorithms. In addition, it is clear that chronological arrivals are beneficial in terms of run-time in comparison with random arrival orderings.

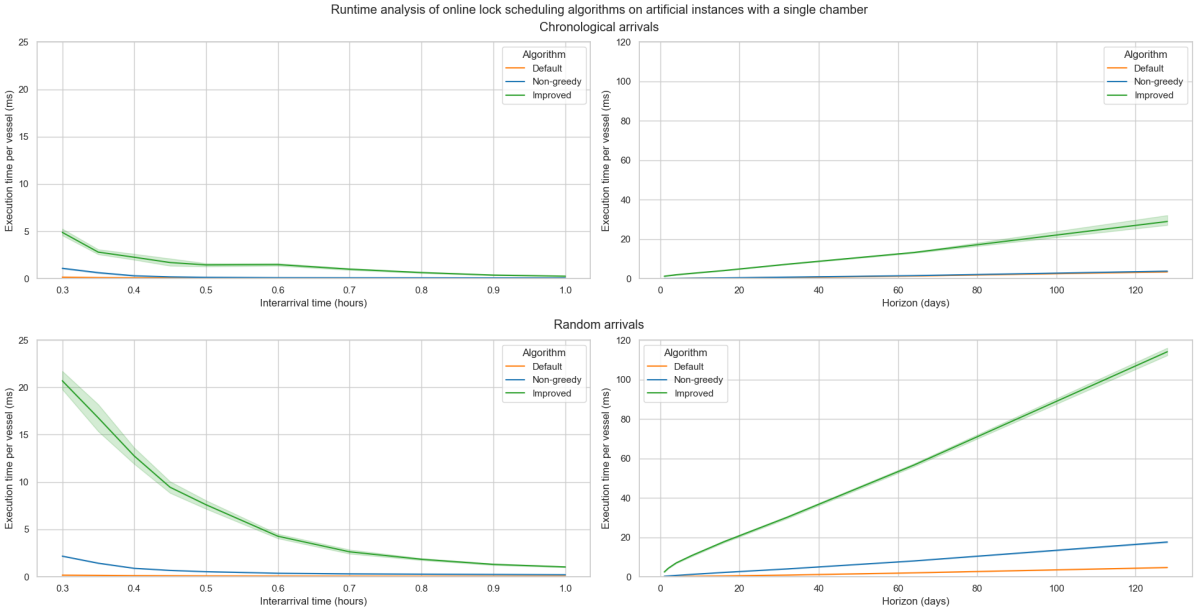


Figure 5.4: Run-time of the online lock scheduling algorithms on artificial instances with a single chamber. Default values for the inter-arrival time and the horizon are 0.6 hours and 2 days respectively.

5.5. Comparison with exact lock scheduling algorithms

Three different algorithms for the unexplored online lock scheduling problem are introduced in the previous sections. Their run-time is analysed compared to each other. However, to determine their ability to create lock schedules of high quality it is important to have a fixed baseline. In this section, an exact offline algorithm is used to find the optimal solution for small problem instances. The algorithms are compared on their relative objective difference with the optimal solution. First, the used exact algorithm and the adaptions made to it are explained. Then, the experimental setup is presented and some considerations regarding the objective function are discussed. The hypothesis is presented before the results are provided.

5.5.1. Exact offline algorithm

Verstichel et al. [8] present a mixed-integer linear program (MILP) to solve the lock scheduling problem with a single lock of multiple chambers to optimality. Ji et al. [26] extend this MILP formulation to include sequential lock movements and perform an interesting comparison between two formulations regarding lock chambers. Verstichel et al. classify chambers by their characteristics (e.g. length, width and lockage duration) into a type. Then lockages are created for each type and each lockage needs to be assigned to a physical chamber. Experiments showed that this makes sense when there are several chambers of the same type but it actually increases the computational complexity of the model

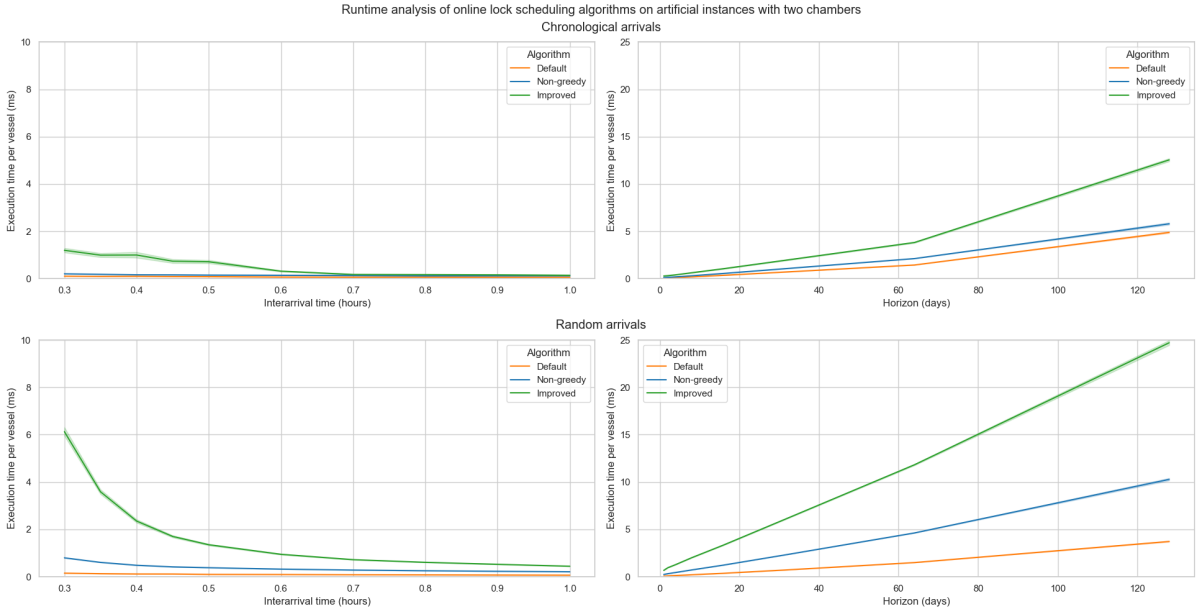


Figure 5.5: Run-time of the online lock scheduling algorithm on artificial instances with two chambers. Default values for the inter-arrival time and the horizon are 0.6 hours and 2 days respectively.

by introducing additional variables if each chamber is unique. In the case of the port of Antwerp, all the locks have different dimensions. Therefore the modifications mentioned by Ji et al. are applied.

As barges are not included in the experiments there is no possibility for vessels to moor onto each other. Therefore the variables ml_{ij} and mr_{ij} which represent that vessel i and j are moored left or right onto each other can be replaced with ml_i. This variable indicates if the vessel is moored onto the left quay of the lock. In the case this is not true, the vessel is moored onto the right quay. This reduces the number of variables from $\mathcal{O}(n^2)$ into $\mathcal{O}(n)$.

In addition, it is required to replace the constraints A.14 - A.31 and A.44 - A.61. Just like in the referenced paper, only the changes for the upstream vessel are presented for conciseness. However, the same applies to the downstream vessels. Equation (5.1) and Equation (5.2) ensure that the x coordinates of the vessels are fixed by the side of the lock they are moored to.

$$x_i \leq (1 - ml_i)W \quad \forall i \in N \tag{5.1}$$

$$x_i + w_i \geq (\sum_{k \in M_t} f_{ik} - ml_i)W_t \quad \forall i \in N, t \in TYPES \tag{5.2}$$

Finally, there are several constraints that can be dropped as they do not apply to the lock scheduling practices assessed in this work. Constraints A.9 and A.39 are omitted because the draught of a vessel is not considered in the lock schedule. The same applies to tide windows. Therefore constraints A.66 & A.67 are omitted. Additionally, it is assumed that the duration of a lockage is constant by the online scheduling algorithms. Constraint A.70 is therefore omitted.

The complete model including downstream ships is added in Appendix A.

5.5.2. Experiment setup

From the historical dataset, a continuously increasing sequence of arrivals is randomly selected until solving the problems becomes infeasible within a practical time limit of 12 hours. For each problem size, 10 instances are created for the exact algorithm. From these instances, 10 additional instances are created for the online lock scheduling algorithms by shuffling the arrival order randomly.

The objective function of the MILP presented by Verstichel et al. only considers the waiting time and the number of lockages containing a vessel. The formulation of Ji et al. does not consider the number of lockages at all. However, in the lock scheduling problem treated in this work, it is considered important to include both filled and empty lockages.

The scheduling part of both formulations is based on the work of Balakrishnan et al. [27]. In this work, it is noticed that other formulations based on the vehicle routing problem could be used to adjust the meaning of seq_{kl} such that it is only set to 1 iif lockage k is directly processed before lockage l. This could be used to determine the number of empty lockages. However, it would also require significantly more variables and constraints to be added. To prevent increasing the complexity of the model further, it is opted to omit counting the empty lockages during this experiment.

During the following experiments, only the summed waiting time objective function is used. The squared waiting time objective function cannot be used as it would destroy the linear properties of the formulation. The different weights for the objective function components are listed in Table 5.6.

Table 5.6: Overview of the different objective weights used in the exact lock scheduling experiments.

Identifier	Total waiting time	Maximum waiting time	Number of lockages
Equal objective	1	1	1
Waiting time only	1	1	0
Realistic objective	1	2	30
Lockage only	0	0	1

5.5.3. Hypothesis

Based on the run-times of the exact algorithm reported in the literature it is expected that the run-time will increase exponentially with an increasing problem size in terms of the number of vessels. In addition, it is expected that the run-time is significantly large compared to the online lock scheduling algorithms. The final hypothesis regarding the exact lock scheduling algorithm on its own is that the problems with the lockage only objective function are solved in less time compared to the other objective functions. This is because it requires only solving the vessel placement sub-problem without actual scheduling. It is possible to let every lockage start after the arrival of the final vessel.

Compared to the online lock scheduling algorithms it is expected that the exact algorithm always finds a solution with an equal or better objective evaluation. Finally, it is expected that the Improved online lock scheduling algorithm finds better solutions compared to the Non-Greedy and Default algorithms.

5.5.4. Results

The experiment with the exact lock scheduling algorithm has been performed on a virtual machine using Ubuntu 20.04.2 LTS, a dual-core processor and 32GB of RAM. The model is implemented in C# using the state-of-the-art CP-SAT hybrid solver from Google OR-Tools [28]. The results of the run-time analysis are listed in Appendix B and support the hypothesis that the algorithm is too slow for online lock scheduling. Within the time limit, the algorithm is able to solve problems up to 10 and 19 vessels for the left and right bank respectively. In addition, the model has also been solved using the Google OR-Tools supported MIP solver, SCIP. However, the observed run-time was higher compared to the CP-SAT hybrid solver.

Figures 5.6 to 5.9 shows the relative objective of the online algorithms compared to the optimal objective value. In addition, it also shows the relative values of every objective component. The first thing to notice is that all but the lockage only objective results in the same solutions. This holds for both the left and the right bank and for every online algorithm. In subsequent experiments comparing the online algorithms by themselves, it will become apparent that the currently used problem instances are too small for differences based on the objective function.

The Non-Greedy and Improved algorithms use fewer lockages compared to the optimal solution. However, their waiting time is higher. Also, the algorithms do not differ in objective function often. This is again attributed to the problem size.

Comparing the algorithms on the problem with a lockage only objective shows that the non-greedy and improved algorithms are good at minimising the number of lockages. This is especially true in the case of two locks.

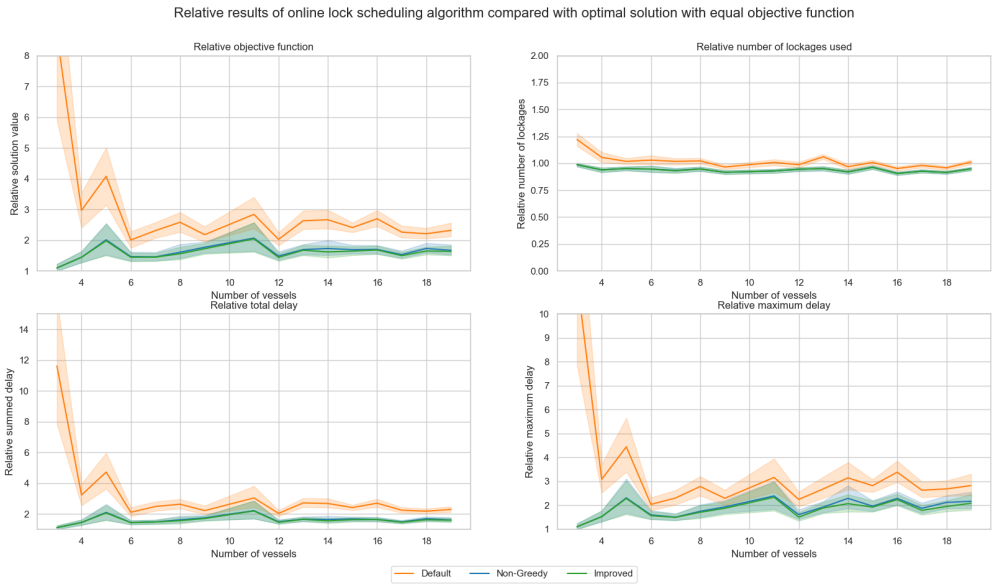


Figure 5.6: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the left bank. The objective function is to minimise the total waiting time, the maximum waiting time and the number of lockages with equal waits.

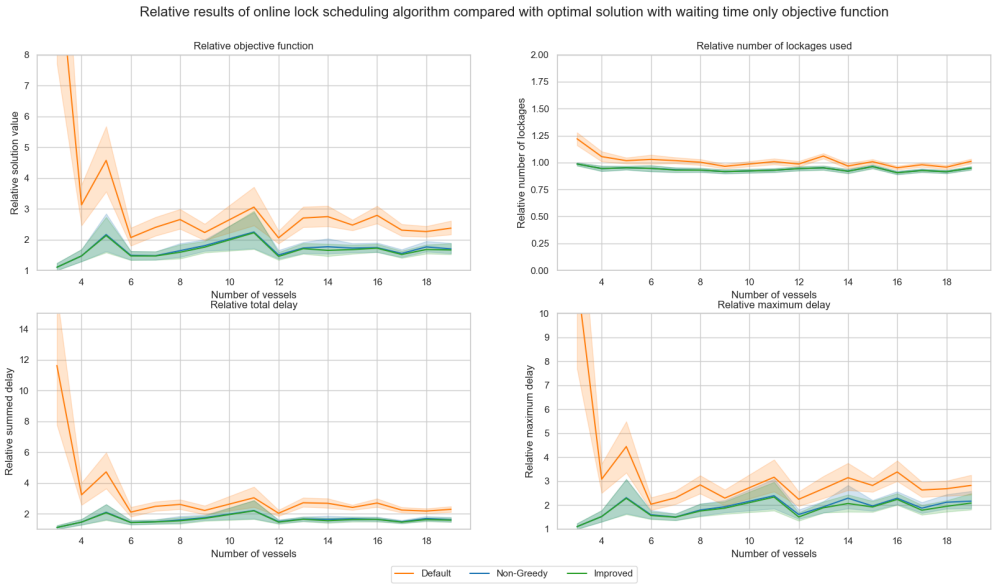


Figure 5.7: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the left bank. The objective function is to minimise the total waiting time and the maximum waiting time.

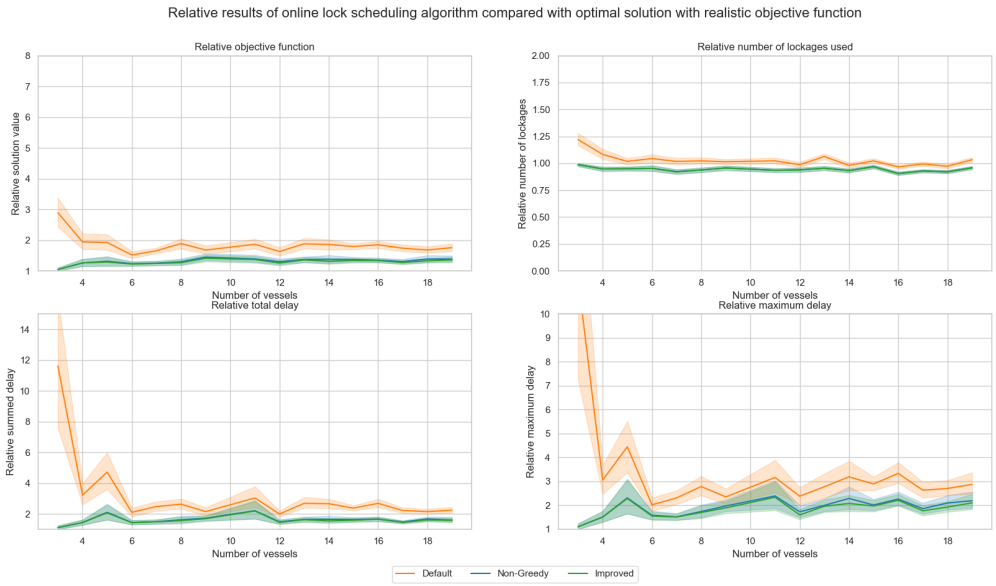


Figure 5.8: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the left bank. The objective function is to minimise the total waiting time, the maximum waiting time and the number of lockages.

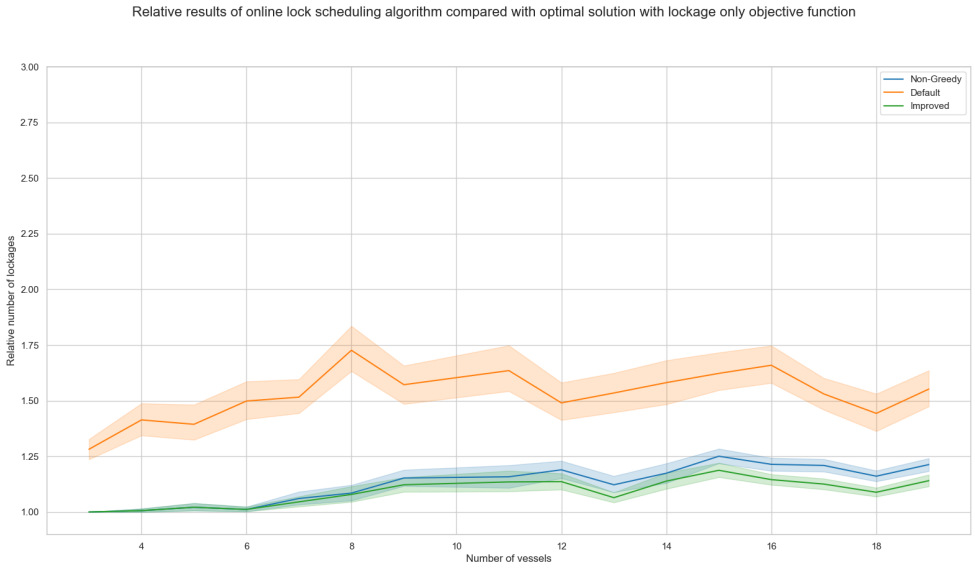


Figure 5.9: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the left bank. The objective function is to minimise the number of lockages.

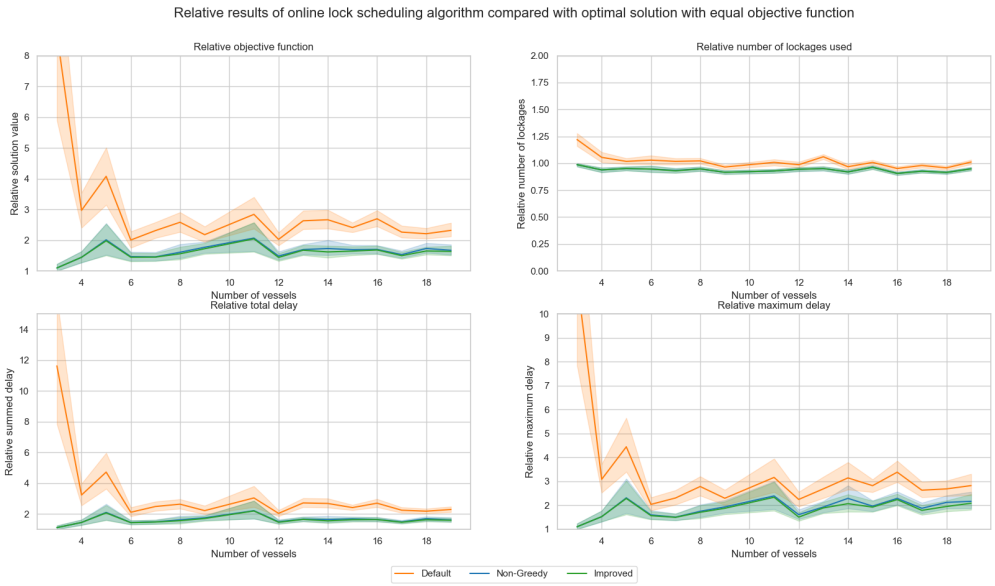


Figure 5.10: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the right bank. The objective function is to minimise the total waiting time, the maximum waiting time and the number of lockages with equal waits.

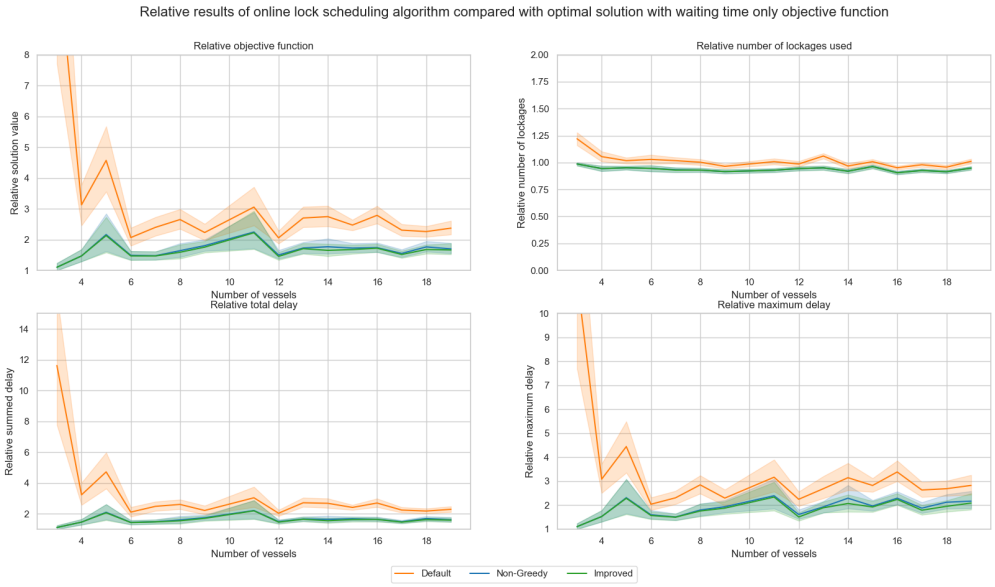


Figure 5.11: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the right bank. The objective function is to minimise the total waiting time and the maximum waiting time.

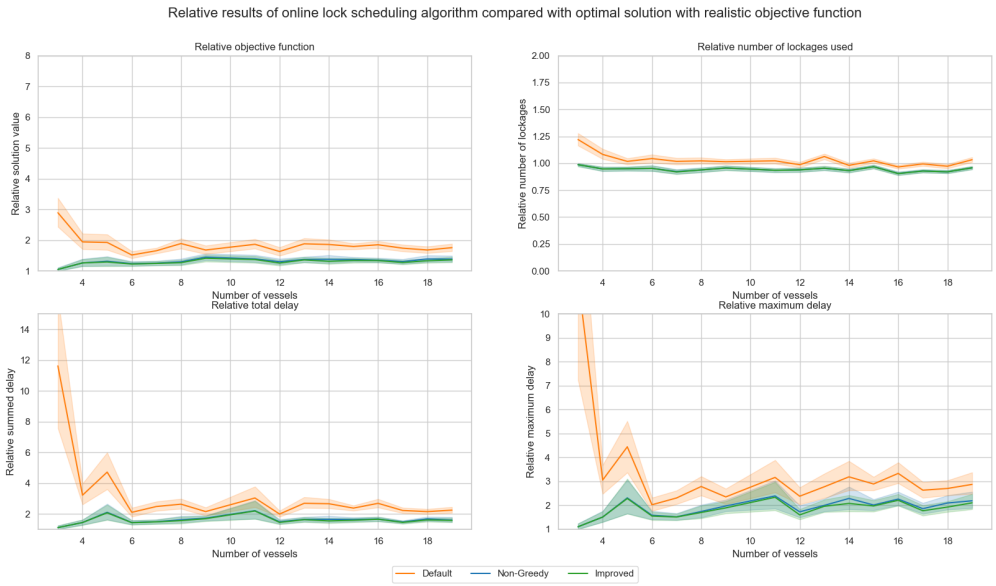


Figure 5.12: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the right bank. The objective function is to minimise the total waiting time, the maximum waiting time and the number of lockages.

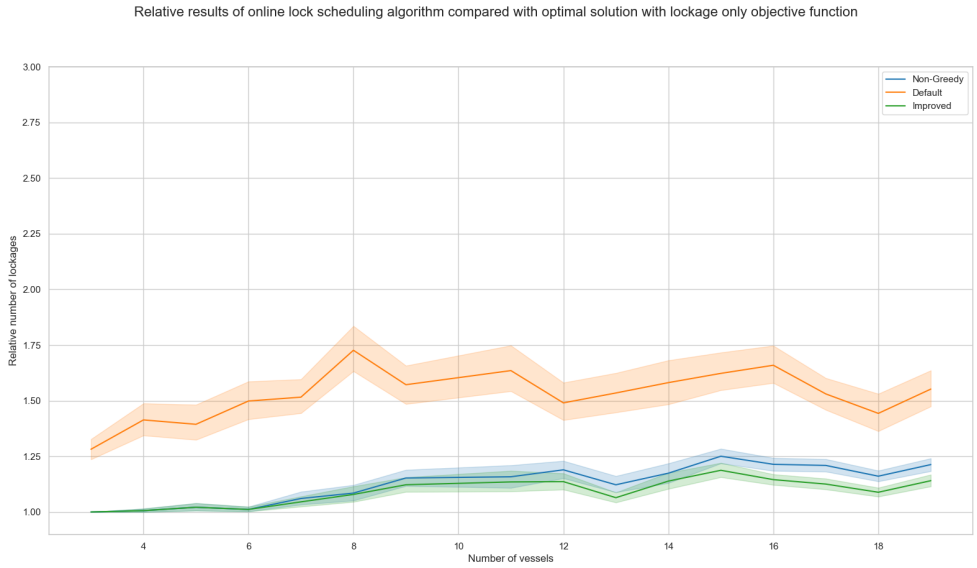


Figure 5.13: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the right bank. The objective function is to minimise the number of lockages.

5.6. Relative lock scheduling comparison

The exact algorithm is not able to solve problem instances of realistic sizes within a reasonable time limit. On these small instances, no significant differences are noticeable between the Non-Greedy and Improved algorithms. Therefore the online lock scheduling algorithms are compared on large problems which the exact algorithm cannot solve. In this section, first, the experimental setup is explained. Then the hypothesis is provided before the results are presented.

5.6.1. Experiment setup

The experiments are performed with 15 different sets of arrivals and departures based on the arrivals of 2019. Each set produces 10 random and one chronological arrival orderings. The length of each

instance is a complete year. Instances are created for each bank.

During the experiments both the summed waiting time and the squared waiting time objective functions are used. For the squared objective function, the weight of the number of lockages is increased. The summed waiting time objective function has two experiments. First, the weight of the maximum waiting time is increased while the weight of the total waiting time and the number of lockages is set to 1 and 30 respectively. Then the weight of the lockages is increased with the weight of the total waiting time set at 1 and the maximum waiting time at 2. Both these objectives are similar to the realistic objective function as introduced during the experiment with the exact algorithm. However, this time the empty lockages are also included.

5.6.2. Hypothesis

Because scheduling vessels in chronological ordering is a good heuristic for a schedule with little waiting time, it is expected that all the algorithms perform better when the arrivals are chronological.

The distinction between the Default and the other algorithm observed during the comparison with the exact algorithm is expected to continue. In addition, due to the larger problem instances, it is expected that the Improved algorithm creates better solutions in comparison with the Non-Greedy algorithm.

Regarding the weights of the objective function components, it is expected that increasing the weight of the number of lockages will reduce them. This will cause an increase in the average and maximum waiting time. In addition, increasing the weight of the maximum waiting time will increase the average waiting time. No effect on the number of lockages is expected as a new lockage is not contributing towards reducing the maximum waiting time.

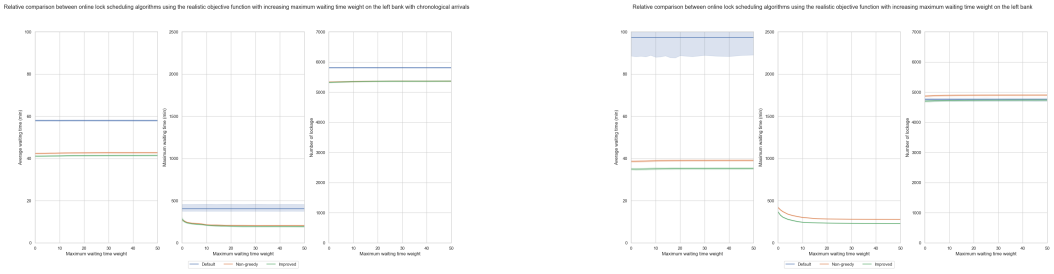
5.6.3. Results

The results of the experiments with the realistic objective function are depicted in Figures 5.14 to 5.17. Increasing the weight of the maximum waiting time causes a small increase in the number of lockages. However, it creates a lock schedule where the maximum waiting time is drastically reduced while the average waiting time remains relatively constant. This trend applies to both banks and the Non-Greedy and Improved algorithms. The Default algorithm results in schedules with extreme maximum waiting times and is not shown in the graph.

Increasing the weight of the number of lockages with the same objective function causes both the average and maximum waiting time to increase. When the number of lockages is drastically reduced compared to the schedule with a low weight, the average waiting time starts to rise quickly. Again, the Default algorithm is not competitive with the other two algorithms in terms of average and maximum waiting time.

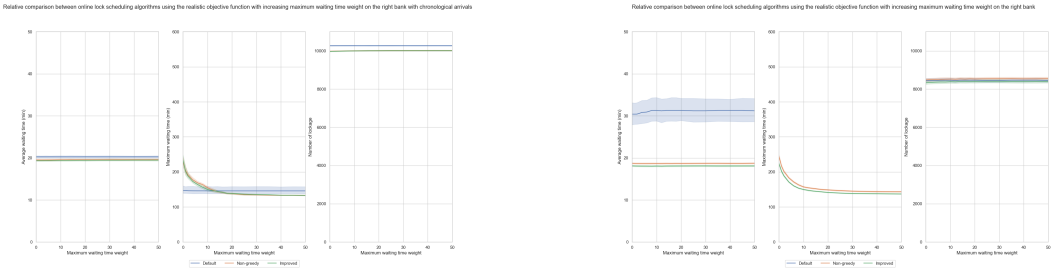
Figures 5.18 and 5.19 shows the results of the squared objective function where the weight of the number of lockages is increased. The same trends as with the realistic objective function are present. However, this time the average waiting time increases harder and the maximum waiting time remains less compared to the other objective function. In addition, the weight of the number of lockages must be relatively high to reach a similar number of lockages as the realistic objective function.

For every objective function, it holds that the Non-Greedy and Improved algorithm have a lower average waiting time and user number of lockages when the ordering of vessels is random. However, the maximum waiting time is larger compared to the chronological ordering. These two algorithms always perform better than the Default algorithm. The Improved algorithm also outperforms the Non-Greedy algorithm. It is especially good at reducing the number of lockages.



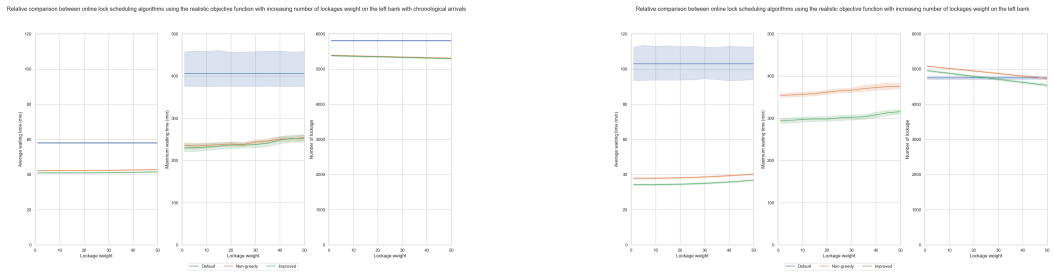
(a) Chronological ordering (b) Random ordering

Figure 5.14: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the left bank where the weight of the maximum waiting time is increased.



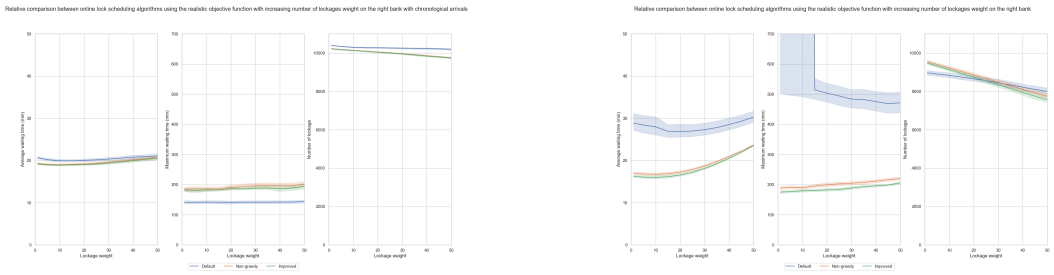
(a) Chronological ordering (b) Random ordering

Figure 5.15: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the right bank where the weight of the maximum waiting time is increased.



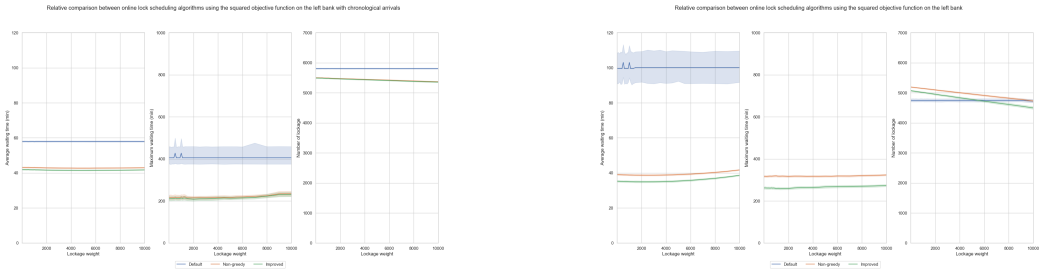
(a) Chronological ordering (b) Random ordering

Figure 5.16: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the left bank where the weight of the number of lockages is increased.



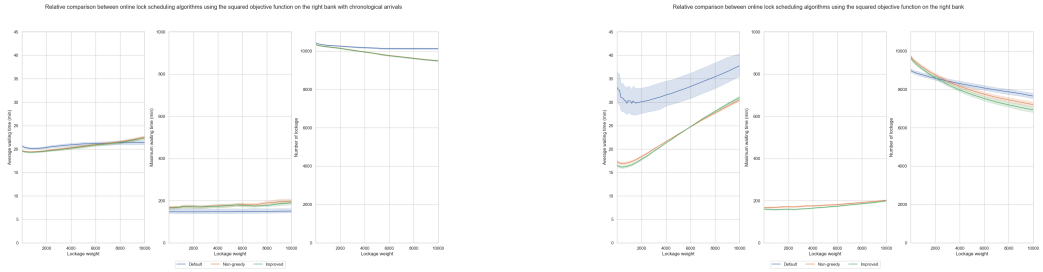
(a) Chronological ordering (b) Random ordering

Figure 5.17: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the right bank where the weight of the number of lockages is increased.



(a) Chronological ordering (b) Random ordering

Figure 5.18: Relative comparison of the online lock scheduling algorithms on realistic instances with an objective function where individual waiting times are squared for the left bank.



(a) Chronological ordering (b) Random ordering

Figure 5.19: Relative comparison of the online lock scheduling algorithms on realistic instances with an objective function where individual waiting times are squared for the right bank.

5.7. Comparison between detailed and abstracted scheduling

In Section 5.1 three online lock scheduling algorithms are presented. It is argued that using a constant lockage duration during scheduling is justified. Arguments for this are the reduced number of interactions with vessels which reduces the complexity of a simulation. Additionally, it is expected that the quality of the lock schedule is not severely affected when converted into a detailed schedule afterwards.

In this section, experiments are performed to evaluate these claims. We first introduce an online lock scheduling algorithm that does not make this simplifying assumption. A method of converting an abstract lock schedule into a detailed lock schedule is presented. Then both detailed and abstracted algorithms will solve problem instances and the results are compared based on the objective function and the number of times interaction with a vessel is required.

5.7.1. Detailed online lock scheduling algorithm

Similar to the abstract online lock scheduling algorithm, the detailed algorithm has to provide different proposals for an arriving vessel. Therefore, the detailed algorithm is based on the same logic as the abstract algorithms. The Improved algorithm is adapted to use the actual lockage duration based on the vessels inside the lockage. This introduces additional complexity to the calculation as it is possible for the first vessel to enter the chamber before the second vessel arrives.

5.7.2. Converting an abstract lock schedule

When scheduling with a constant lockage duration, the algorithms often create lockages that will take more time in reality. Therefore an abstract schedule must be converted into a detailed schedule. A simplistic approach, which does no additional optimisation, is the following push-back method.

1. Create an empty detailed lock schedule.
2. Starting at the first lockage in the abstract schedule, the actual lockage duration is determined based on the vessels allocated.

3. Based on the predecessor in the detailed lock schedule and the vessels inside the lockage, the earliest start-time of the lockage is determined.

4. A new lockage is added to the detailed lock schedule with the updated start time and duration.

5. The conversion continues with the next lockage in the abstract schedule at step 3.

Note that using this method, lockage gets pushed back towards a later start time if a lockage turns out to use more than the time anticipated. However, the contrary is also true; when a lockage takes less time than scheduled, the sequential lockage could start earlier if the vessels are able to arrive at that moment.

5.7.3. Experiment setup

During the experiments, both the abstract and detailed online lock scheduling algorithms schedule all the arrivals. The schedule of the abstract algorithms is converted according to the push-back method. Each vessel requires 10 minutes to enter and another 10 minutes to exit the lock in the final detailed schedules. Instances are based on three months of historical arrivals and one month of generated arrivals for 2030.

25 instances with random arrival orderings are generated per bank and year of arrivals in addition to chronological arrivals. The constant lockage duration is varied for the abstract algorithms. Each instance is solved using three objective functions. The realistic and waiting time only objective functions are selected from the experiment with the exact algorithm. In addition, the squared waiting time objective of the relative comparison is used with a weight of 900 for the number of lockages. This is roughly similar to 30 minutes additional delay, depending on the number of vessels getting delayed. The objective of optimising solely the number of lockages is omitted as it is not realistic and does not vary depending on the constant lockage duration.

Because the detailed schedule contains a starting time of a vessel, a vessel could arrive before another vessel is able to reach the lock. It is therefore not possible to use the start time of a lockage as the reference for the waiting time. During these experiments, the waiting time is replaced with tardiness.

During each step of the process of creating a detailed lock schedule, it is counted how often a vessel would have been questioned. The interactions are counted during both the evaluation of every proposal, the execution of the selected proposal and during the conversion of an abstract schedule into a detailed one. During the following operations, the number of vessels inside that lockage is counted.

1. Adding a vessel to a lockage

2. Changing the start time of a lockage

3. Changing the duration of a lockage

5.7.4. Hypothesis

Based on the earlier made assumption, it is expected that the abstract algorithms require significantly fewer interactions while still providing results of similar quality. As mentioned, the objective function is not a perfect measurement of the quality of a lock schedule. This is especially true when squaring individual waiting times. A small increase in the total waiting time could result in a significant objective difference. It is therefore that besides the objective function, this comparison is also made based on the average and maximum waiting time combined with the number of lockages.

5.7.5. Results

Figures 5.20 and 5.21 show the relative objective, its components and the number of interactions of the abstract algorithms compared to the detailed algorithm when using the realistic objective function. The improved abstract algorithms performs best and therefore its results are listed. The optimal relative objective is achieved with a constant lockage duration between 50 and 55 minutes. It reduces the interactions with between 32 and 40% while increasing the average waiting time with 11 and 15%. Figures 5.22 and 5.23 and Figures 5.24 and 5.25 show the same results when using the squared and waiting time only objective functions respectively. Table 5.7 summaries the results and shows the relative differences for the optimal constant lockage duration.

The trends are similar for every objective function. When the constant lockage duration increases the relative total and maximum tardiness increases. However, the number of lockages decreases. This is because when scheduling lockages with a long duration, more vessels can be added to it. Interestingly, even in the case that the number of lockages was not included, the abstract algorithms tend to use fewer lockages compared to the detailed algorithm.

Table 5.7: Overview of the optimal constant lockage duration, the relative number of interactions and the different objective components for different objective functions with the improved abstract algorithm.

Objective function	Optimal constant lockage duration (min)	Relative objective function	Relative number of interactions	Relative average tardiness	Relative maximum tardiness	Relative number of lockages
Realistic	50-56	1.03-1.05	0.60-0.68	1.10-1.15	1.13-1.30	0.89-0.92
Squared	54-58	1.12-1.17	0.61-0.68	1.08-1.13	1.08-1.21	0.94-0.97
Waiting time only	44-48	1.05-1.07	0.58-0.65	0.98-1.02	1.00-1.15	0.93-0.97

5.8. Summary

This chapter introduced a framework for online lock scheduling algorithms. Three algorithms based upon this framework are presented and compared on their run-time and solution quality. The assumption that a constant lockage duration reduces the interactions while having a limited impact on the lock schedule quality is validated.

There are differences in the run-time of the algorithms. However, each of them shows the same trends with a varying inter-arrival time and plannings horizon. In addition, the run-time remains within a few milliseconds per arrival. This would be a negligible part of a simulation for the complete Port of Antwerp.

Compared to an offline exact lock scheduling algorithm, the online algorithms are extremely fast and competitive on the lock schedule quality. Often, the algorithms result in fewer lockages and a slightly higher average and maximum waiting time.

The Improved algorithm outperforms the other two algorithms on realistic problem instances with different objective functions. The Default algorithm is by no means competitive with the other two algorithms. Different objective functions result in lock schedules with different characteristics.

Experiments with both a constant and a detailed lockage duration show a predictable reduction in interactions per constant lockage duration. There is a small increase in the average tardiness and a decrease in the number of lockages. Especially, the maximum tardiness increases when using a constant lockage duration. The experiments also show that none of the objective functions is a perfect proxy of the lock schedule quality.

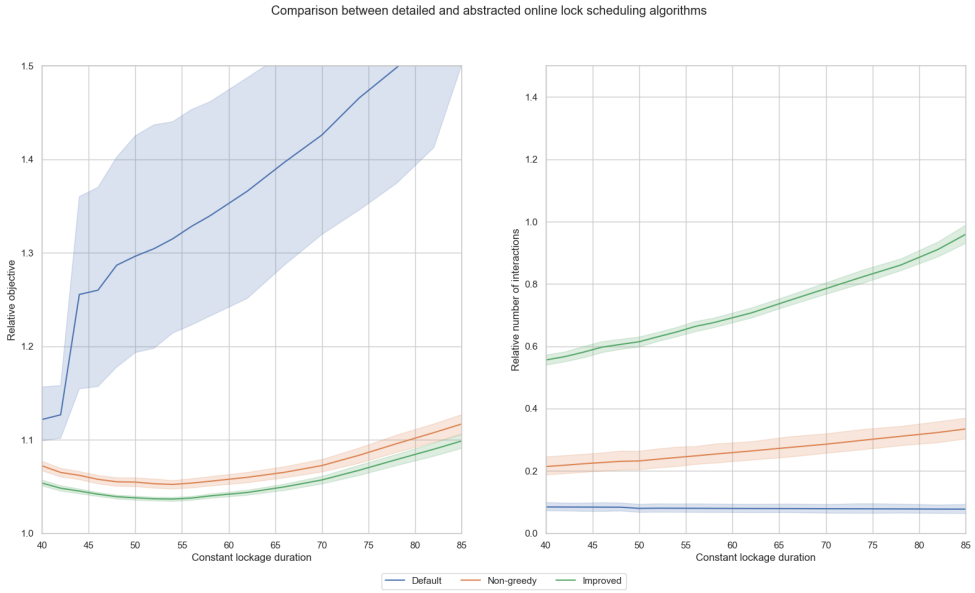


Figure 5.20: Relative objective and fraction of interactions required for the abstract algorithms compared to the detailed online lock scheduling algorithm using the realistic objective function.

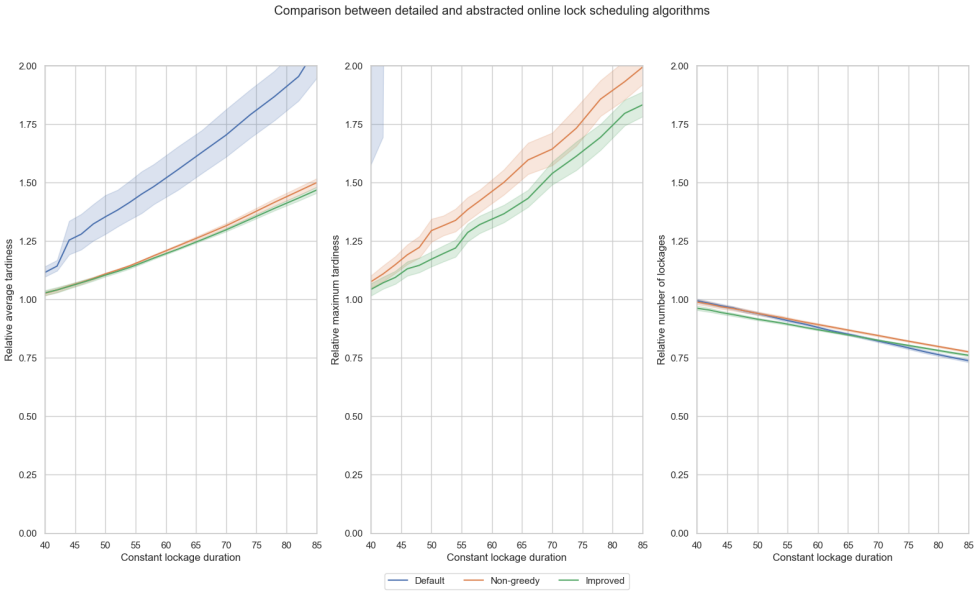


Figure 5.21: Relative objective components of the resulting schedules of the abstract algorithms compared to the detailed online lock scheduling algorithm using the realistic objective function.

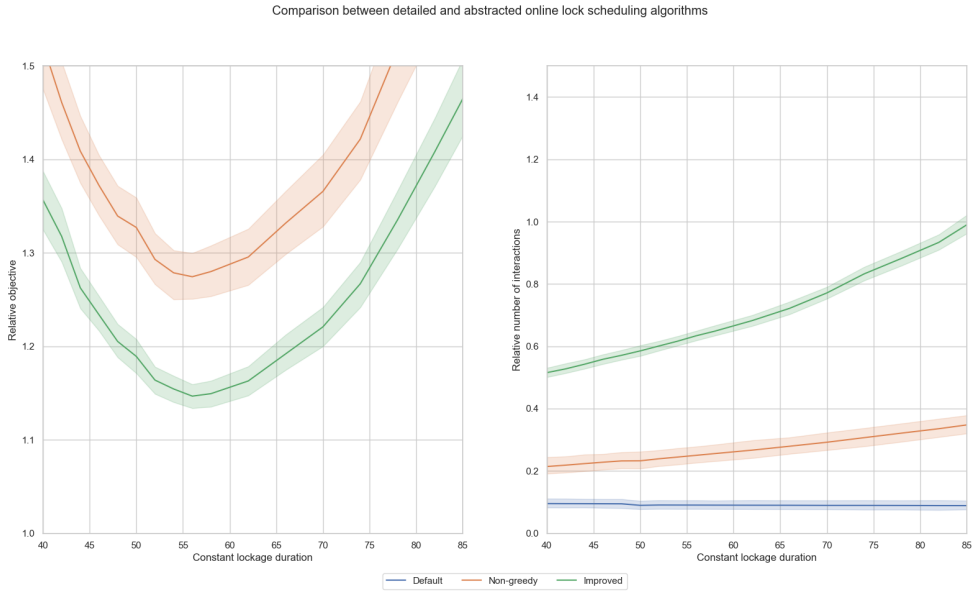


Figure 5.22: Relative objective and fraction of interactions required for the abstract algorithms compared to the detailed online lock scheduling algorithm using the squared objective function.

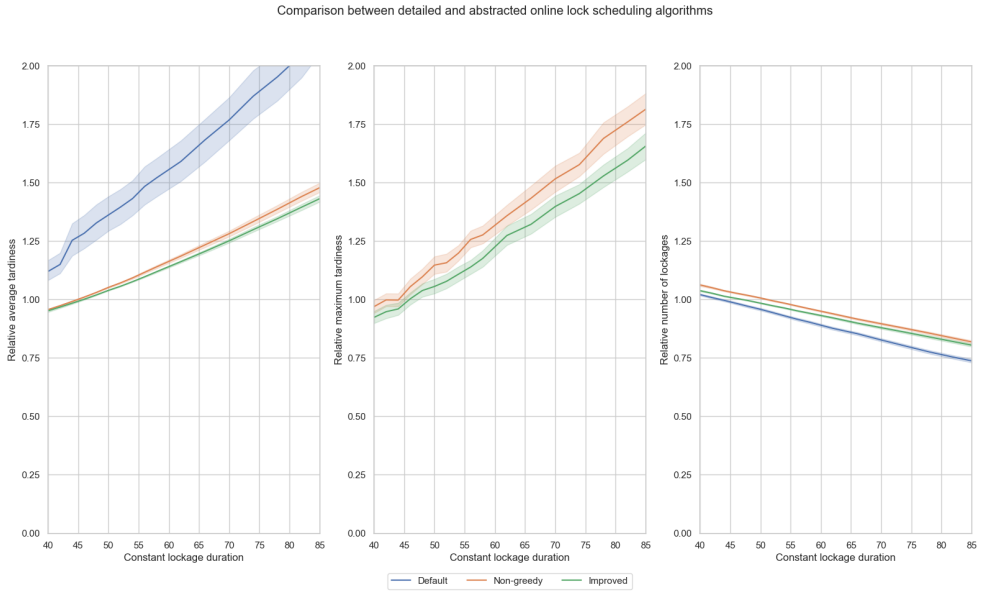


Figure 5.23: Relative objective components of the resulting schedules of the abstract algorithms compared to the detailed online lock scheduling algorithm using the squared objective function.

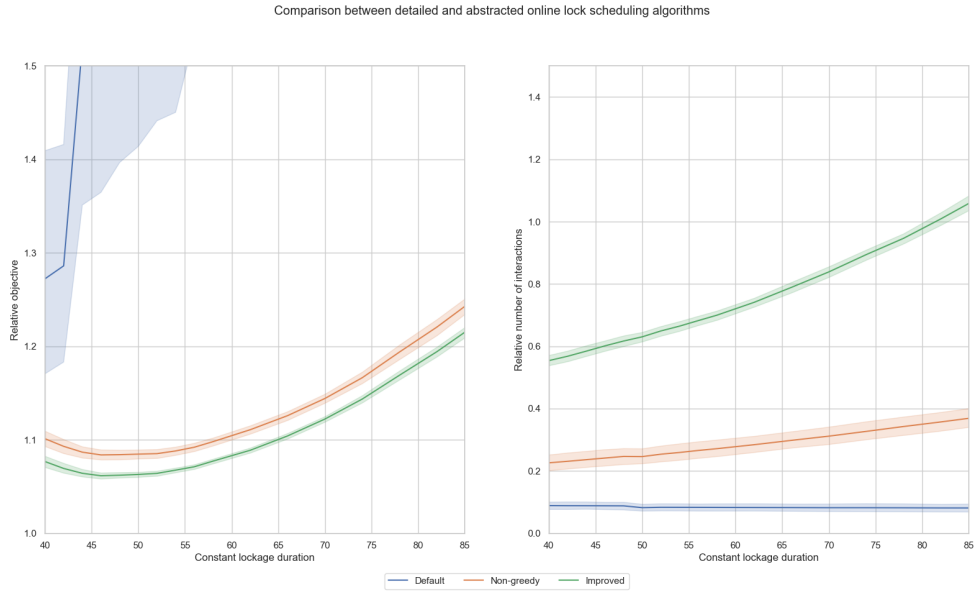


Figure 5.24: Relative objective and fraction of interactions required for the abstract algorithms compared to the detailed online lock scheduling algorithm using the squared waiting time only function.

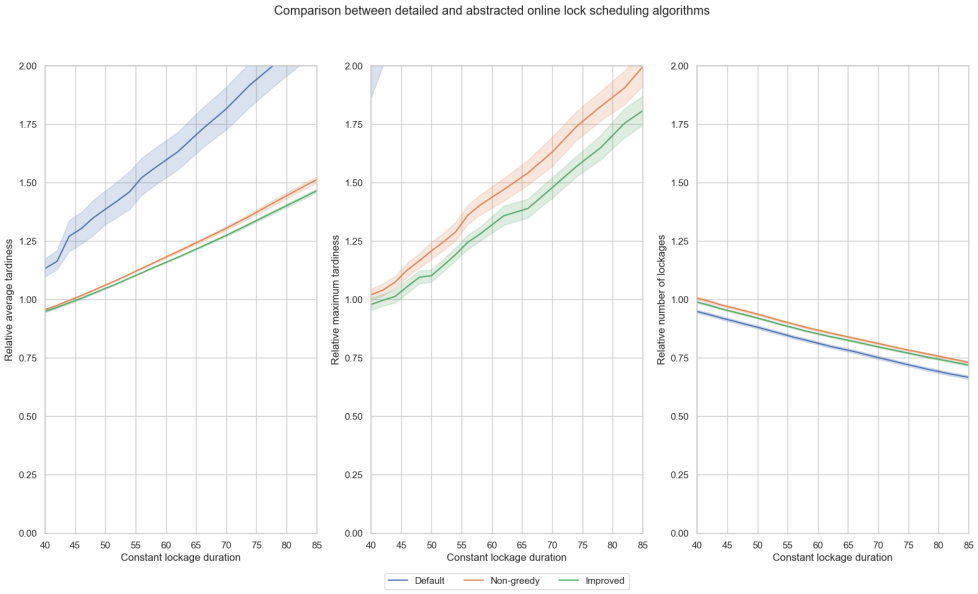


Figure 5.25: Relative objective components of the resulting schedules of the abstract algorithms compared to the detailed online lock scheduling algorithm using the waiting time only objective function.

6

Disruption management

A disruption occurs when a vessel is not able to arrive at the moment it is scheduled in the initial schedule. This chapter first introduces the problem related to disruption management and the available mitigations. Then, the selected algorithmic approaches to solve the problem are explained. Different metrics to define the quality of a recovery are defined before the results are presented.

6.1. The disruption management problem

Vessels try to reach the locks at exactly the moment they are planned. However, they may not always succeed. This section explains the possible types of delays and tries to provide a ballpark in which they might occur. In addition, examples of the possible mitigations are provided. All information in this section is provided by lockmasters at the Port of Antwerp.

The most occurring delay of vessels is between 5 and 10 minutes. This is within the 15 minutes buffer between lockages. Therefore it won't cause severe issues for the schedule. In addition, vessels never arrive earlier than anticipated as they can always travel slower along their route.

In the rare cases that a vessel is delayed for a longer duration, significant rescheduling could be required. There are several possible actions to take. However, all of them will introduce additional waiting times to vessels. First of all, barges have less priority compared to vessels. It is therefore possible to remove barges from the lockage to ensure it can start as soon as possible as the delayed vessel arrives. Complementary, a delay for departing vessels is preferred over arriving vessels. This is because arriving vessels are likely already travelling towards the port while departing vessels only leave their berth shortly before entering the lock.

When delaying the lockage of the delayed vessel will cause too much delay for other vessels or subsequent lockages, it is possible to move the vessel to a later lockage. However, in such cases, it must be ensured the vessel can be delayed that much in addition to its current delay. It is therefore sometimes possible to delay the entire existing schedule for a few minutes as the delayed vessel is not able to wait for the next lockage.

6.2. Algorithms

This section introduces how the in chapter 5 introduced online lock scheduling algorithms can be used for disruption management. Then the different neighbourhood search algorithms for lock scheduling are assessed on their applicability for disruption management. The most promising of them is selected and extended to focus the search around the disruption.

6.2.1. Online lock scheduling algorithms

During the comparison between abstract and detailed online lock scheduling, it was explained how the abstract algorithms could operate with a detailed schedule. It is required for disruption management to not assume a constant lockage duration and allocated specific arrival times for each vessel.

Each of the three abstract algorithms is converted and used for disruption management. This is achieved by removing the disrupted vessel from the existing schedule and rescheduling it with its up-

dated arrival time. The Non-Greedy and Improved algorithms can delay other lockages. Therefore they can execute a right-shift of the entire schedule when no other feasible solution can be found. In addition, the Improved algorithm can perform a small local search using the local improvements already defined.

6.2.2. Neighbourhood search

The existing online lock scheduling algorithms are limited in their search for better lock schedules because every change must be related to the disrupted vessel. Neighbourhood search techniques can explore a larger part of the search space. It has been successfully applied to disruption management for airlines. Similar algorithms exist to create an initial lock schedule.

Neighbourhood search consists of creating an initial solution and adapting it continuously to find better solutions. This is similar to disruption management where the initial solution consists of the disrupted lock schedule. Table 6.1 shows the different neighbourhood search algorithms applied to lock scheduling together with the characteristics of the resulting schedule. An algorithm that has all three properties can be directly integrated.

Table 6.1: Overview of the research regarding neighbourhood search techniques for lock scheduling and the properties of their resulting schedule. More properties of the resulting schedule available is better.

Authors	Year	Vessel allocation	Heterogeneous chambers	Non-constant lockage duration
Ji et al. [13]	2019	yes	yes	yes
Prandtstetter et al. [14]	2015	no	no	yes
Verstichel and Vanden Berghe [16]	2009	yes	yes	no
Verstichel et al. [15]	2011	no	yes	no

Based on the table, one can easily observe that only the algorithm of Ji et al. fulfils all the requirements of a detailed lock schedule. This adaptive large neighbourhood search (ALNS) algorithm is selected and consists of several destroy and repair operations which are also applicable for other problems. A destroyed lock schedule is repaired by scheduling each removed vessel using the Improved detailed lock scheduling algorithm.

As the online algorithms are known to result in sub-optimal lock schedules it is expected that the ALNS requires additional help to focus on the disruption. Therefore, the following additional destroy operations are proposed;

1. **Related arrivals and directions:** there already exists an operation that selects a random vessel and then continues with finding the most similar vessels based on their arrival time. The introduced operation only selects vessels in the same direction to guide the search towards solving the disruption with only affecting lockages in the same direction.

2. **Related to disruption:** both the introduced destroy operation and the operation it is based on select a random vessel to start with. This makes sense when creating an initial schedule where optimisations can be achieved everywhere. However, during disruption management, it is desired to only affect the minimum required number of vessels. Therefore two additional destroy operations that use the same logic to select related vessels but start with the disrupted vessel are introduced.

6.3. Objective function

While solving the disruption it is important to keep the objective function similar to the one used during scheduling. This is to prevent changing the schedule significantly. In addition, a balance between the objectives of the scheduling algorithms and the number of vessels affected is required. Similar to the experiment where abstract and detailed lock scheduling is compared, it is required to use tardiness instead of waiting time during these experiments because of the detailed lock schedules.

By definition, a vessel is affected when both the start and end of the lockage it is placed is is earlier than initially planned. In addition, when a lockage finishes later it is also affected. To discourage affecting a large number of vessels more severe than a few, the number of affected vessels is squared.

Another approach to reduce the number of vessels affected and to focus the search on the disruption is penalising affecting vessels far from the disruption. This large penalty is applied when a vessel, more than 24 hours after the disruption, gets affected. In addition, to reduce the search space where such large penalties are given, the input for the algorithm is cut-off at 48 hours after the disruption.

6.4. Experiment design

Because the lock schedule is created by the online lock scheduling algorithms, it is not possible to compare the performance of the disruption resolution based on the objective function. The ALNS is expected to find some changes to the schedule that could improve the objective function while not contributing to resolving the disruption. Therefore, the lockage containing the disrupted vessel in the final schedules of the online algorithms and the ALNS is compared. When there is a difference in the vessels, start and/or end time of the lockage, this is called a disagreement.

Each experiment starts with an initial schedule created by one of the online lock scheduling algorithms. These algorithms are both the detailed and the abstract version of the online lock scheduling algorithms. For the latter, the resulting schedules are converted into a detailed schedule using the push-back method. A constant lockage duration of 60 minutes is used. All the experiments are performed using the summed waiting time objective function with the weights of the realistic variant as introduced by the comparison of the online lock scheduling algorithms and the exact algorithm. Because the hyper-parameters could be different for other objective functions only a single objective function is evaluated.

A smaller inter-arrival time allows the algorithms to change more vessels to resolve the disruption. The instances of the disruption management experiments are therefore based on generated arrivals for 2030. The length of these schedules is irrelevant because of the cut-off after 48 hours. A vessel is randomly selected from the initial schedule and delayed for a predefined duration. This ranges from two until 40 minutes with increments of two minutes.

These instances are used to determine the effect of the disruption duration and the initial scheduling algorithm on the quality of the resolution. Because the disrupted vessel is randomly selected, it is possible to reuse the same schedule a few times. 11 schedules are created and from each schedule 10 vessels are randomly selected. Each algorithm will solve 720 instances per disruption duration and 2400 instances per initial algorithm.

In addition to the realistic problem instances, instances are generated similarly as for the run-time analysis of the online lock scheduling algorithms. This time, only the inter-arrival time is of interest. It ranges from 0.3 until 0.55 hours with 0.05 hour increments. The resulting schedules contain significantly more vessels per hour compared to the realistic instances. This is relevant for the case when traffic increases and barges are added to the simulation. Again every online lock scheduling algorithm is used to create the initial schedule. The disruption duration range from 15 until 25 minutes with increments of 5 minutes.

The next section explains the hyper-parameter tuning of the ALNS. The optimal parameters found during this process are used during the experiments. First, the lock scheduling objective function is used to solve the disruptions on the realistic and generated problem instances. Then the number of affected vessels is added to the objective function to determine if it helps to focus the ALNS on the disruption.

6.5. Hyper-parameter tuning

The paper introducing the ALNS for lock scheduling provides the optimal parameters for the algorithm. However, given that the problem at hand is slightly different and also changes to the objective function are made, it is not possible to use them straight away. This section describes the parameters that might need to be adjusted and the method used to find the optimal value.

Because of the different objective functions and problems at hand, it is reasonable to expect that parameters related to the objective could be improved. Therefore the initial temperature, which is defined as a fraction of the initial objective, and the cooling rate are selected for the hyper-parameter tuning.

The number of iterations per segment and the number of segments are also related to the objective function and how fast it converges. However, the run-time of the algorithm is still, with a few seconds, relatively small. In addition, early experiments showed that the objective converged well before the

final iteration is reached. Therefore, these parameters are not included in the hyper-parameter tuning.

The maximum number of vessels the ALNS is allowed to destroy in a single iteration has initially been set to 50. However, there are fewer arrives per day. This implies that the whole schedule would be renewed. As this is not the goal of disruption management, smaller values are evaluated during the hyper-parameter tuning.

An overview of all the parameters of the ALNS and its default values are presented in Table 6.2. It also contains the range of the parameters subject to the hyper-parameter tuning and the final selected values. Due to the limited number of parameters, it is possible to use a grid search among all parameters combinations. Experiments with these parameters are performed on problem instances generated in a similar manner as explained in the previous section. However, to prevent over-fitting the historical arrivals of 2019 are used.

Table 6.2: Overview of the parameters of the adaptive large neighbourhood search with the original value provided by Ji et al. [13] In addition, the range of values evaluated during hyper-parameter tuning and the selected values are listed when applicable.

Parameter	Default	Range	Selected
Iterations per segment	50	-	-
Number of segements	20	-	-
Maximum degree of disruption	50	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	5
Initial temperature	0.2	[0.001, 0.01, 0.1, 0.2]	0.001
Cooling rate	0.995	[0.9, 0.95, 0.995]	0.995
Reaction factor	0.05	-	-
K1	0.0	-	-
K2	1.0	-	-
K3	0.1	-	-

Based on the number of disagreements there is no difference between any parameter combination. However, when optimising the objective functions, the ALNS performs better with the minimal initial temperature. With a higher temperature, it accepts worse solutions too often to find better solutions. In addition, when increasing the maximum degree of disruption the algorithm tends to find better solutions. However, this parameter was added to the hyper-parameter tuning to prevent rescheduling all the vessels in the schedule. Therefore it is set to 5, after which the largest decline in the objective is reached. The cooling rate did not influence the quality of the resulting lock schedule and is therefore set to the default value.

6.6. Hypothesis

If the ALNS can provide an advantage over the online lock scheduling algorithms, it is expected that it often disagrees with them. In addition, it is expected that including the number of affected vessels in the objective function increases the number of disagreements as the search should be focused on the already affected vessel. Finally, assuming that the ALNS is providing the expected advantage over the online lock scheduling algorithms, it is expected that decreasing the inter-arrival time will increase the number of disagreements. This is because the shorter inter-arrival time causes vessels to be placed together in lockages. This allows for more optimisation where the ALNS is expected to excel.

6.7. Results

The effect of the disruption duration on the disagreements between the online lock scheduling algorithms and the ALNS on the realistic problem instances is depicted in Figure 6.1. It shows a sharp increase in disagreements for the Default algorithm after a duration of 15 minutes is reached. This is the default buffer between lockages. Its essential working becomes clear as it requires little to no rescheduling. The other online algorithms have close to no disagreements with the ALNS.

The results for the generated problem instances are depicted in Figure 6.3. The first graph shows the number of disagreements between the ALNS and the online lock scheduling algorithms for a variable inter-arrival time. Although the inter-arrival time is higher on realistic instances, it shows the increase in disagreements with a decreasing inter-arrival time. The second graph has a fixed inter-arrival time

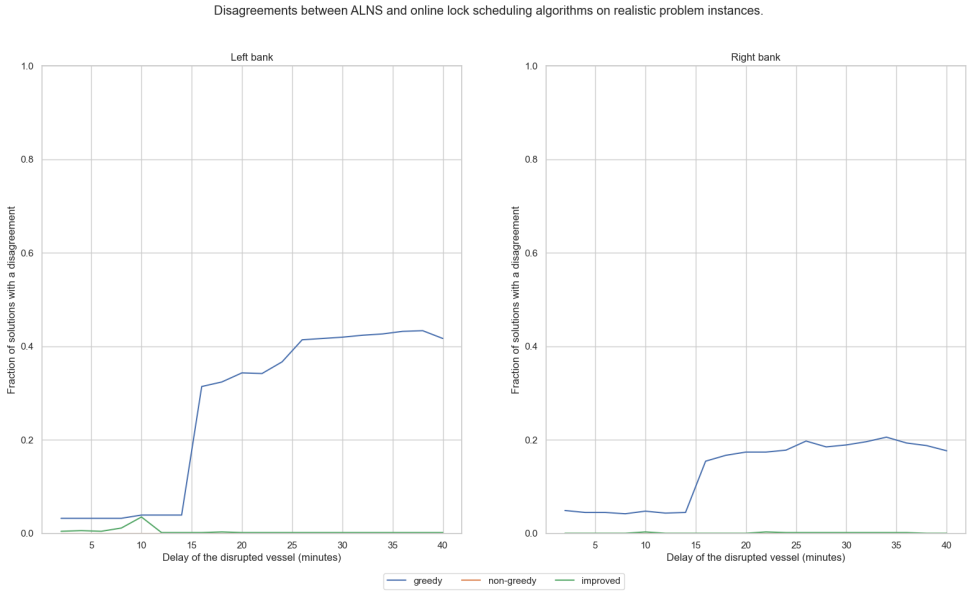


Figure 6.1: Fraction of solutions with a disagreement between online lock scheduling algorithms and the adaptive large neighbourhood search on realistic problem instances with different disruption durations and all algorithms used to create the initial schedule on instances.

of 0.5 hours. It shows that including the number of affected vessels in the objective function decreases the number of disagreements.

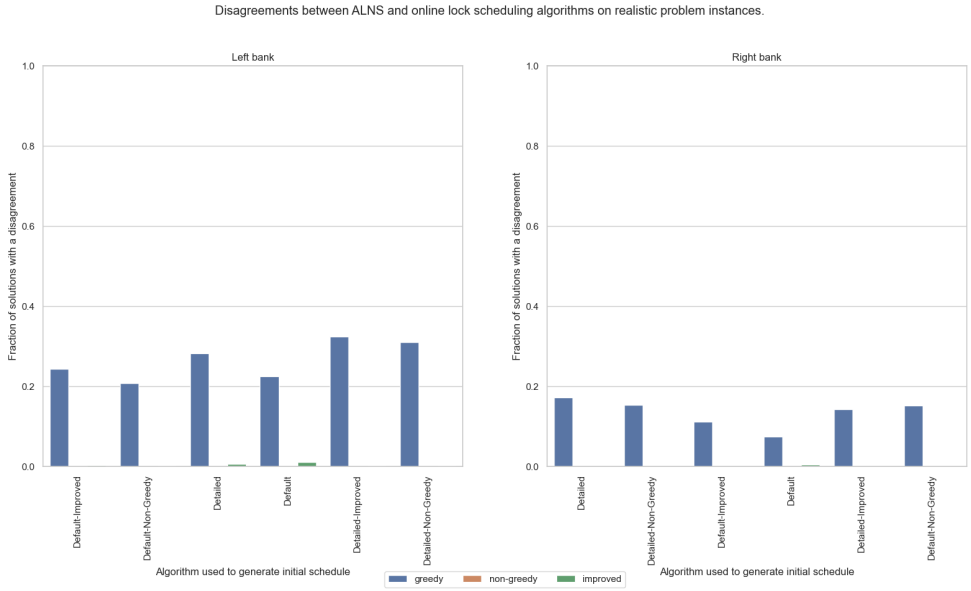


Figure 6.2: Fraction of solutions with a disagreement between online lock scheduling algorithms and the adaptive large neighbourhood search on realistic problem instances with different algorithms used to create the initial schedule on instances with all disruption durations.

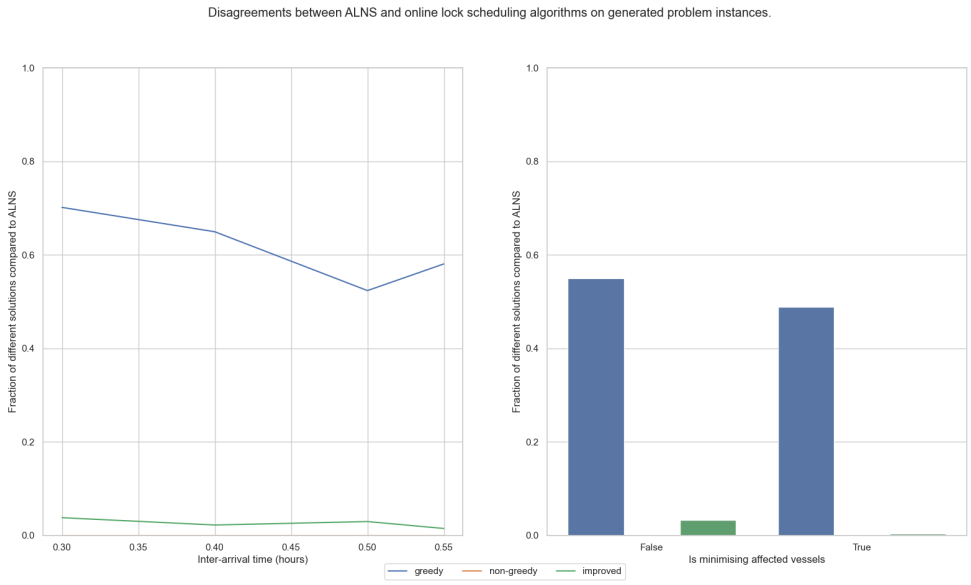


Figure 6.3: Fraction of disagreements between online lock scheduling algorithms and the adaptive large neighbourhood search on generated problem instances. Left shows the results for a variable inter-arrival time. Right shows the difference between including the number of affected vessels in the objective function with a fixed inter-arrival time of 0.5 hours.

7 Discussion

This chapter provides an overview of the performed experiments and discusses their results. The different online lock scheduling algorithms are summarised before assessing the experiments comparing their performance. Then, the trade-off between lock schedule quality and the number of interactions is discussed. Finally, the experiments regarding disruption management are considered.

7.1. Lock scheduling algorithms and results

In chapter 5, three algorithms for the online lock scheduling problem were introduced. Each of these algorithms operates in a framework of generating proposals for the arriving vessel, selecting the best proposal and executing it. They vary in their ability to affect already-scheduled vessels and lockages. The Default algorithm is not able to change the position of scheduled vessels and is thus forced to make greedy decisions. Both the Non-Greedy and the Improved algorithms are allowed to come back to their decision and change the position of already scheduled vessels. The Non-Greedy algorithm is able to delay other vessels in order to create a new lockage. This is improved upon by the Improved algorithm by means of a local search where proposals are improved. All three algorithms use the multi-order best-fit heuristic to determine an allocation for all the vessels inside the locks chamber.

The algorithms are intended to be used in a simulation where a vessel plans its journey through multiple schedules. Changing the position of vessels requires the other schedules to check feasibility again. To reduce the number of interactions and thus the complexity of the simulation, a constant lockage duration is used during scheduling.

The introduced algorithms are compared to an exact offline lock scheduling algorithm in section 5.5. This algorithm has an advantage over the online algorithms because it knows all the arrivals from the start. The comparison could only be performed on small problem instances due to the extended run-time of the exact algorithm. On these instances the Non-Greedy and Improved algorithms are competitive. The number of lockages used is close and often slightly less compared to the optimal solution. In addition, the average and maximum waiting-time are doubled compared to the optimal solution. No significant differences between the right and left banks are observed.

In the previous comparison, it already became clear that being able to affect already planned vessels provides an advantage. The Default algorithm performed less compared to the others even on very small problem instances. However, there was little to no difference between the Non-Greedy and Improved algorithms. Except when solely minimising the number of lockages. Therefore the algorithms are compared relative to each other in section 5.6.

The Improved online lock scheduling algorithm outperforms the other algorithms on all three objective functions. The Default algorithm only comes close to the other algorithms when the vessels arrive in chronological ordering. This shows that scheduling in chronological ordering is a fairly good heuristic. Being able to come back to previously made actions and correct them later is especially useful when the ordering is random. The Non-Greedy and Improved algorithm result in schedules with fewer lockages and a lower average waiting time. However, the maximum waiting time increases compared to the chronological ordering.

Overall, the Improved algorithm performs similar on small problems and better on large instances

compared to the Non-Greedy algorithm. The Default algorithm is by no means competitive with the other algorithms.

7.2. Trade-off between interactions and lock schedule quality

During scheduling, a constant lockage duration is used to reduce the number of interactions with vessels. After the schedule is completed, it gets converted into a detailed lock schedule using a push-back method. The online lock scheduling algorithms do not directly optimise for the final objective function due to this operation. The experiments in section 5.7 analyse how this affects the final lock schedule and how many interactions are avoided using this method.

The results show that the number of reduced interactions is relatively constant based on the used constant lockage duration. The optimal constant lockage duration varies based on the objective function. However, it remains below the duration of a lockage with 2 vessels.

Table 5.7 summarises the results for the optimal constant lockage duration for the abstract Improved algorithm. It shows that the relative objective function is a poor proxy for the actual metrics. This holds especially for the squared objective function. The number of interactions is reduced by one third when selecting the optimal constant lockage duration. This increases the average and maximum waiting time by approximately 10%. When excluding the number of lockages from the objective function the average waiting time becomes roughly equal.

Overall, it is concluded that using a constant lockage duration simplifies creating the lock schedule while having a small impact on the quality of the produces schedule.

7.3. Disruption management

Chapter 6 applied the online lock scheduling algorithms for the problem of disruption management. The algorithms are compared to an adaptive large neighbourhood search meta-heuristic. This meta-heuristic has originally been designed to create initial lock schedules but could solve disruptions with minimal adaptions. Additional destroy operators are introduced that are focused on solving the disruption instead of global optimisation.

Because of the sub-optimal schedules created by the online lock scheduling algorithms and the tendency of the meta-heuristic to optimise the complete schedule, it is not possible to compare the algorithms for disruption management based on the objective function. Even when the algorithms resolve the disruption equally, the objective function could be different if an improvement is found somewhere in the lock schedule. As a proxy for the quality of the resolution of the disruption the position of the disrupted vessel is compared. If the meta-heuristic finds a different position for the disrupted vessel it is assumed to be a better resolution.

The results presented in section 6.7 show that the meta-heuristic always finds the same position for the disrupted vessel as the Non-Greedy algorithm. Especially when the inter-arrival time of the vessels is low, the Improved algorithm disagrees with the meta-heuristic. The Improved algorithm proposes the same actions as the Non-Greedy together with some extra locally improved actions. Therefore, the Improved algorithm always finds equal or better actions. When the meta-heuristic agrees with the Non-Greedy algorithm but disagrees with the Improved algorithm, it is assumed that the latter has found the best resolution.

Overall, it is concluded that the Non-Greedy and Improved online lock scheduling algorithms are competitive with ALNS based on the position of the disrupted vessel. However, it must be noted that the used metric is not a perfect proxy for the quality of the resolution.

8

Conclusion

The research questions posed in the introduction are answered in this chapter. The first section answers the sub-questions before answering the main research question. Then research directions that could further improve the presented work are discussed.

8.1. Research questions

The posed sub-questions regarding lock scheduling and their answers are as following;

- **How can the lock scheduling problem posed by the Port of Antwerp be defined as an optimisation problem?**

  Chapter 2 introduced the generalised lock scheduling problem where the arrivals of all vessels are known from the beginning and the possible objective function components. It continued with an adaption to accommodate the use-case of the Port of Antwerp. This implies that vessels are gradually added to the schedule in no particular ordering.

- **What algorithms can solve the defined lock scheduling problem?**

  In chapter 3, the existing solution methods for variants of the lock scheduling problem were discussed. Particular variants with strict and limiting assumptions about the locks and vessels have exact polynomial-time algorithms. Both exact mixed-integer linear programs and heuristics have been applied to less restricted problems. Run-times of the exact algorithms are unpredictable and too large, even for small problem instances. In addition, none of the existing methods was designed for incrementally expanding an existing lock schedule.

  A framework for online lock scheduling algorithms is introduced in section 5.1. The procedure consists of three steps; First, a number of possible actions to accommodate the vessel are created. Then the actions are evaluated based on the objective function. Finally, the best action is performed on the existing schedule. In the same section, three different algorithms are presented which use this framework to solve the online lock scheduling problem. They vary in their capabilities of affecting vessels once they are planned.

- **How can problem abstraction during scheduling contribute to fewer interactions with vessels?**

  Section 5.7 introduced a method to convert a lock schedule where every lockage has a constant duration into a detailed schedule with a duration based on the vessels placed in a lockage. Two versions of the online lock scheduling algorithms are compared on the number of interactions they require and the quality of the resulting lock schedule for different objective functions. One version uses a constant lockage duration while the other determines the duration based on the vessels in the lockage. It shows that an appropriate constant lockage duration reduces the interactions significantly with a small effect on the average and maximum tardiness.

The posed sub-questions regarding disruption management and their answers are as following;

- **What algorithms can be used for disruption management?**

  The literature review showed in chapter 3 that the disruption management problem is unexplored in the context of lock scheduling. Section 6.2 explains how the detailed version of the online lock scheduling algorithms can be used to solve this problem. In addition, the most suitable algorithm from the available neighbourhood search algorithms in the literature is selected and adapted for disruption management. These algorithms have in common that they remove one or more vessels from an existing schedule and add them again using standard lock scheduling techniques.

- **Which parameters do affect the quality of the recoveries?**

  Defining the quality of a disruption resolution depends on both the number of vessels affected and the additional delay and lockages in the updated lock schedule. However, it is impossible to just use these metrics to evaluate a recovery. This is because disruption unrelated operations could be performed. This work compares the position of the disrupted vessel in the resolution of different algorithms. There is no pattern in the number of disagreements for the different initial lock scheduling algorithms. When the disruption duration is smaller than the buffer between lockages, even the Default lock scheduling algorithms agrees often with the other algorithms. Decreasing the inter-arrival time of vessels causes an increase in the number of disagreements.

With all the sub-questions answered, the main research questions can be answered;

Which solution methods prove to be the most effective to automate the lock scheduling and real-time disruption recovery for the Port of Antwerp?

It can be concluded that the Improved online lock scheduling algorithm using a constant lockage duration is the most suitable for the Port of Antwerp. This algorithm results in lock schedules with the best objective of all the compared algorithms. While its run-time is higher, it is still within a few milliseconds per vessel. Finally, the constant lockage duration helps to reduce the number of interactions with vessels.

The Detailed online lock scheduling algorithm with detailed lockage durations is competitive with the adaptive large neighbourhood search for disruption management based on the lockage of the disrupted vessel. Due to the unnecessary global optimisation and additional run-time of the meta-heuristic, it can be concluded that the online algorithm is more suitable. The improved version of the online algorithms only suggests additional possibilities of resolving the disruption. It can therefore also be used.

8.2. Future research directions

This section identifies six directions future research could take to improve or built upon the presented work. Five of them are detailed and focused on creating the initial lock schedule. The final research direction is broader and is related to the disruption management problem.

- **Adding barges to the arrivals:** Through the period in which this thesis is created, no data was available about the arrivals of barges. These types of vessels have lower priority and are planned within a shorter horizon. However, the large number of them is expected to affect the algorithms. The different experiments could be repeated with the barges included to determine their impact and if the conclusions still hold.

- **Better optimality gap determination:** The exact algorithm used to determine the optimal lock scheduling algorithm is not able to solve large problem instances to optimality within a reasonable amount of time. In addition, the used objective function is limited as it does not include empty lockages. To determine the performance of the online lock scheduling algorithm compared to an objective baseline, other algorithms are required. When an exact algorithm turns out to be insufficient, meta-heuristics like the adaptive large neighbourhood search could be used.

- **Additional local search operators:** The Improved algorithms utilise two local search operators to improve upon the Non-Greedy algorithm. Future research could focus on finding additional operators and determine their individual contributions. Especially search operators comparing lockages of different chambers are unexplored.

- **Integration with realistic port simulation:** During the experiments it was assumed that a vessel can always be delayed. This is because there were no other schedules around the lock schedules. To determine the actual effect of scheduling with a constant lockage duration it would be good to include other realistic schedules which must cooperate.
- **Trade-off due to lock scheduling abstractions:** In this work the trade-off between interactions with vessels and lock schedule quality caused by using a constant lockage duration is explored. Another commonly made abstraction is regarding the capacity of a locks chamber and replaces the vessel placement sub-problem with a number of vessels. It is interesting to determine if a similar approach of dealing with this abstraction exists. In addition, the trade-off could also be made with offline algorithms.
- **Using different meta-heuristics for disruption management:** The online lock scheduling algorithms are comparative with an adaptive large neighbourhood search meta-heuristic for disruption management. As these algorithms are rather simple it is likely that some meta-heuristic could improve upon them.

Bibliography

[1] B. Ji et al. “Optimally solving the generalized serial-lock scheduling problem from a graph-theory-based multi-commodity network perspective”. In: *European Journal of Operational Research* 288.1 (2021), pp. 47–62. DOI: 10.1016/j.ejor.2020.05.035.

[2] J. Verstichel et al. “Exact and heuristic methods for placing ships in locks”. In: *European Journal of Operational Research* 235.2 (2014). Maritime Logistics, pp. 387–398. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2013.06.045.

[3] J. Verstichel and G.V. Berghe. *Scheduling serial locks: A green wave for waterbound logistics*. 2015, pp. 91–109. DOI: 10.1007/978-3-319-17419-8_5.

[4] Jens Hermans. “Optimization of inland shipping”. In: *Journal of Scheduling* 17.4 (2014), pp. 305–319.

[5] Ward Passchyn et al. “The lockmaster’s problem”. In: *European Journal of Operational Research* 251.2 (2016), pp. 432–441. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2015.12.007.

[6] Ward Passchyn, Dirk Briskorn, and Frits C. R. Spieksma. “No-Wait Scheduling for Locks”. In: *INFORMS Journal on Computing* 31.3 (2019), pp. 413–428. DOI: 10.1287/ijoc.2018.0848.

[7] Ward Passchyn and Frits C.R. Spieksma. “Scheduling parallel batching machines in a sequence.” In: *Journal of Scheduling* 22.3 (June 2019), pp. 335–357. ISSN: 1094-6136. DOI: 10.1007/s10951-018-0560-6.

[8] Jannes Verstichel et al. “The generalized lock scheduling problem: An exact approach”. In: *Transportation Research Part E: Logistics and Transportation Review* 65 (2014). Special Issue on: MODELING, OPTIMIZATION AND SIMULATION OF THE LOGISTICS SYSTEMS, pp. 16–34. ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2013.12.010.

[9] J. Verstichel et al. “A Combinatorial Benders’ decomposition for the lock scheduling problem”. In: *Computers & Operations Research* 54 (2015), pp. 117–128. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2014.09.007.

[10] Bin Ji et al. “An exact approach to the generalized serial-lock scheduling problem from a flexible job-shop scheduling perspective”. In: *Computers & Operations Research* 127 (2021), p. 105164. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2020.105164.

[11] Bin Ji et al. “Optimally solving the generalized serial-lock scheduling problem from a graph-theory-based multi-commodity network perspective”. In: *European Journal of Operational Research* 288.1 (2021), pp. 47–62. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.05.035.

[12] Wouter Van Adrichem. “LOSCO A Ship Lock Scheduling Model”. MA thesis. Delft University of Technology, 2020. URL: http://resolver.tudelft.nl/uuid:ed6b3620-1e73-47b0-9600-4a71a73de670.

[13] Bin Ji et al. “An adaptive large neighborhood search for solving generalized lock scheduling problem: Comparative study with exact methods”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.8 (2019), pp. 3344–3356.

[14] Matthias Prandtstetter et al. “A Variable Neighborhood Search Approach for the Interdependent Lock Scheduling Problem”. In: *Evolutionary Computation in Combinatorial Optimization*. Ed. by Gabriela Ochoa and Francisco Chicano. Cham: Springer International Publishing, 2015, pp. 36–47. ISBN: 978-3-319-16468-7.

[15] Jannes Verstichel, Patrick De Causmaecker, and Greet Vanden Berghe. “Scheduling algorithms for the lock scheduling problem”. In: *Procedia - Social and Behavioral Sciences* 20 (2011). 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, pp. 806–815. ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2011.08.089.

[16] Jannes Verstichel and Greet Vanden Berghe. “A Late Acceptance Algorithm for the Lock Scheduling Problem”. In: Jan. 2009, pp. 457–478. ISBN: 978-3-7908-2361-5. DOI: 10.1007/978-3-7908-2362-2_23.

[17] Jinfen Zhang et al. “Sequential ship traffic scheduling model for restricted two-way waterway transportation”. In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 231.1 (2017), pp. 86–97. DOI: 10.1177/1475090215621580.

[18] Andy M. Ham and Eray Cakici. “Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches”. In: *Computers & Industrial Engineering* 102 (2016), pp. 160–165. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2016.11.001.

[19] Sungbum Jun, Seokcheon Lee, and Hyonho Chun. “Learning dispatching rules using random forest in flexible job shop scheduling problems”. In: *International Journal of Production Research* 57.10 (2019), pp. 3290–3310. DOI: 10.1080/00207543.2019.1581954.

[20] Marcella Samà et al. “Coordination of scheduling decisions in the management of airport airspace and taxiway operations”. In: *Transportation Research Procedia* 23 (2017). Papers Selected for the 22nd International Symposium on Transportation and Traffic Theory Chicago, Illinois, USA, 24-26 July, 2017., pp. 246–262. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2017.05.015.

[21] Yi Su et al. “Airline Disruption Management: A Review of Models and Solution Methods”. In: *Engineering* 7.4 (2021), pp. 435–447. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2020.08.021.

[22] V. Subramaniam, A. Raheja, and K. Reddy. “Reactive repair tool for job shop schedules”. In: *International Journal of Production Research* 43 (Jan. 2005), pp. 1–23. DOI: 10.1080/00207540420002704l2.

[23] R. J. Abumaizar and J.A. Svestka. “Rescheduling job shops under random disruptions”. In: *International Journal of Production Research* 35.7 (1997), pp. 2065–2082. DOI: 10.1080/002075497195074.

[24] Velusamy Subramaniam and AmritpalSingh Raheja. “mAOR: A heuristic-based reactive repair mechanism for job shop schedules”. In: *The International Journal of Advanced Manufacturing Technology* 22 (Jan. 2003), pp. 669–680. DOI: 10.1007/s00170-003-1601-6.

[25] Shinji Imahori and Mutsunori Yagiura. “The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio”. In: *Computers & Operations Research* 37.2 (2010), pp. 325–333. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2009.05.008.

[26] Bin Ji et al. “An exact approach to the generalized serial-lock scheduling problem from a flexible job-shop scheduling perspective”. In: *Computers & Operations Research* 127 (Mar. 2021), p. 105164. DOI: 10.1016/j.cor.2020.105164.

[27] Nagraj Balakrishnan, John J. Kanet, and V. Sridharan. “Early/tardy scheduling with sequence dependent setups on uniform parallel machines”. In: *Computers & Operations Research* 26.2 (1999), pp. 127–141. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(98)00051-3.

[28] Laurent Perron and Vincent Furnon. *OR-Tools*. Version 9.0. Google, Aug. 20, 2021. URL: https://developers.google.com/optimization/.

A

Mixed Integer Linear Program Formulation

A.1. Parameters

N, N': Set of upstream, downstream vessels, indexed by i, j.
M, M': Set of the upstream, downstream lockages, indexed by k, l.
w_i, l_i: Width and length of upstream vessel i.
w_i', l_i': Width and length of downstream vessel i.
dF_i, dB_i: Minimal distance between upstream vessel i and the front, back of the chamber.
dF_i', dB_i': Minimal distance between downstream vessel i and the front, back of the chamber.
sL_{ij}: Minimal safety distance between upstream vessels i and j when they are lying behind each other.
sL_{ij}': Minimal safety distance between downstream vessels i and j when they are lying next to each other.
sW_{ij}: Minimal safety distance between upstream vessels i and j when they are lying behind each other.
sW_{ij}': Minimal safety distance between downstream vessels i and j when they are lying next to each other.
r_i, r_i': Arrival time of upstream, downstream vessel i at the coordination point.

U: Set of physical chambers indexed by u.
W_u, L_u: Width and length of the chamber u.
W, L: Maximal width and length over all chambers
M_u, M_u': Subset of M, reserved for upstream, downstream lockages performed on chamber u.
p: Constant lockage duration.
$setup_{kl}$: Minimal setup time between lockages k and l when they are processed by the same chamber. Depends on the direction of lockages k and l.
C_{max}: Big M constant used as an upper bound for the completion time.

A.2. Variables

$\mathbf{x_i}$, $\mathbf{y_i}$: Integer variables that define the x and y position of vessel i (front left corner).
$\mathbf{b_{ij}}$: Binary variable indicating whether vessel $i$is to the left of vessel j or not.
$\mathbf{left_{ij}}$: Binary variable indicating whether vessel i is behind vessel j or not.
$\mathbf{ml_i}$: Binary variable indicating whether vessel i is moored onto the left side of the chamber or not.
$\mathbf{z_k}$: Binary variable that indicates whether lockage k is used or not.
$\mathbf{f_{ik}}$: Binary variable that indicates whether vessel i is processed in lockage k or not.
$\mathbf{v_{ij}}$: Binary variable that indicates whether vessels i and j are processed in the same lockage or not.
$\mathbf{c_i}$: Departure time of vessel i.
$\mathbf{C_k}$: Completion time of lockage k.
$\mathbf{seq_{kl}}$: Binary variable that indicated whether lockage k precedes lockage l or not.
$\mathbf{T_{max}}$: Maximum waiting time of all vessels.

A.3. Objective function

The values of K_1, K_2 and K_3 are variable and are defined for each experiment performed with the algorithm.

$$K_1 \sum_{k \in N \cup N'} (c_i - p - r_i) + K_2 T_{max} + K_3 \sum_{k \in M \cup M'} Z_k \tag{A.1}$$

A.4. Constraints

The following blocks of constraints models the scheduling part of the lock scheduling problem.

$$c_i \geq C_{max}(f_{ik} - 1) + C_k, \qquad \forall i \in N, k \in M \tag{A.2}$$
$$c_i \leq C_{max}(1 - f_{ik}) + C_k, \qquad \forall i \in N, k \in M \tag{A.3}$$
$$c_i' \geq C_{max}(f_{ik} - 1) + C_k', \qquad \forall i \in N', k \in M' \tag{A.4}$$
$$c_i' \leq C_{max}(1 - f_{ik}) + C_k', \qquad \forall i \in N', k \in M' \tag{A.5}$$

$$p_k \geq pz_k, \qquad \forall k \in M \tag{A.6}$$
$$p_k \geq pz_k, \qquad \forall k \in M' \tag{A.7}$$

$$C_l - C_k \geq p_l + setup_{kl} - 2C_{max}(1 - seq_{kl}), \qquad \forall k < l \in M_u \cup M_u', u \in U \tag{A.8}$$
$$C_k - C_l \geq p_k + setup_{kl} - 2C_{max}(1 - seq_{kl}), \qquad \forall k < l \in M_u \cup M_u', u \in U \tag{A.9}$$

$$C_k - P_k \geq f_{ik} r_i, \qquad \forall i \in N, k \in M \tag{A.10}$$
$$C_k - P_k \geq f_{ik} r_i, \qquad \forall i \in N', k \in M' \tag{A.11}$$

$$Z_k \leq \sum_{i \in N} f_{ik}, \qquad \forall k \in M \tag{A.12}$$
$$Z_k \leq \sum_{i \in N'} f_{ik}, \qquad \forall k \in M' \tag{A.13}$$

The following block of constraints are transitive constraints and are used to break symmetry.

$$Z_{k+1} \leq Z_k, \qquad \forall k \in M_u \cup M_u', u \in U \tag{A.14}$$
$$C_k \leq C_{k+1}, \qquad \forall k \in M_u \cup M_u', u \in U \tag{A.15}$$

The following constraint is used for the objective function.

$$T_{max} \geq c_i - r_i - p, \qquad \forall i \in N \cup N' \tag{A.16}$$

The following blocks of constraints models the ship placement part of the lock scheduling problem for the upstream vessels.

$$left_{ij} + left_{ji} + b_{ij} + b_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geq 1, \qquad \forall i < j, i, j \in N, k \in M \tag{A.17}$$
$$x_i - x_j + W left_{ij} \leq W - w_i, \qquad \forall i, j \in N \tag{A.18}$$
$$y_i - y_j + L b_{ij} \leq L - l_i, \qquad \forall i, j \in N \tag{A.19}$$
$$x_j - x_i + (W + sW_{ij})(1 - left_{ij} + b_{ij}) \geq w_i + sW_{ij}, \qquad \forall i, j \in N \tag{A.20}$$
$$y_j - y_i + (L + sL_{ij})(1 - b_{ij} - left_{ij}) \geq l_i + sL_{ij}, \qquad \forall i, j \in N \tag{A.21}$$
$$x_i + w_i \leq W_u + (1 - f_{ik})W, \qquad \forall i \in N, k \in M_u, u \in U \tag{A.22}$$
$$y_i + l_i \leq L_u + (1 - f_{ik})L, \qquad \forall i \in N, k \in M_u, u \in U \tag{A.23}$$
$$y_i \geq dF_i, \qquad \forall i \in N \tag{A.24}$$
$$y_i + l_i \leq Lu - dB_i + (1 - f_{ik}), \qquad \forall i \in N, k \in M_u, u \in U \tag{A.25}$$

$$\sum_{k \in M} f_{ik} = 1, \qquad \forall i \in N \tag{A.26}$$

$$f_{ik} \leq Z_k, \qquad \forall i \in n, k \in M \tag{A.27}$$

$$x_i \leq (1 - ml_i)W, \qquad \forall i \in N \tag{A.28}$$

$$x_i + w_i \geq (\sum_{k \in M_u} f_{ik} - ml_i)W_u, \qquad \forall i \in N, u \in U \tag{A.29}$$

The following blocks of constraints models the ship placement part of the lock scheduling problem for the downstream vessels.

$$left_{ij} + left_{ji} + b_{ij} + b_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geq 1, \qquad \forall i < j, i, j \in N', k \in M' \tag{A.30}$$

$$x_i - x_j + Wleft_{ij} \leq W - w_i, \qquad \forall i, j \in N' \tag{A.31}$$

$$y_i - y_j + Lb_{ij} \leq L - l_i, \qquad \forall i, j \in N' \tag{A.32}$$

$$x_j - x_i + (W + sW_{ij})(1 - left_{ij} + b_{ij}) \geq w_i + sW_{ij}, \qquad \forall i, j \in N' \tag{A.33}$$

$$y_j - y_i + (L + sL_{ij})(1 - b_{ij} - left_{ij}) \geq l_i + sL_{ij}, \qquad \forall i, j \in N' \tag{A.34}$$

$$x_i + w_i \leq W_u + (1 - f_{ik})W, \qquad \forall i \in N', k \in M'_u, u \in U \tag{A.35}$$

$$y_i + l_i \leq L_u + (1 - f_{ik})L, \qquad \forall i \in N', k \in M'_u, u \in U \tag{A.36}$$

$$y_i \geq dF_i, \qquad \forall i \in N \tag{A.37}$$

$$y_i + l_i \leq Lu - dB_i + (1 - f_{ik}), \qquad \forall i \in N', k \in M'_u, u \in U \tag{A.38}$$

$$\sum_{k \in M'} f_{ik} = 1, \qquad \forall i \in N' \tag{A.39}$$

$$f_{ik} \leq Z_k, \qquad \forall i \in n, k \in M' \tag{A.40}$$

$$x_i \leq (1 - ml_i)W, \qquad \forall i \in N' \tag{A.41}$$

$$x_i + w_i \geq (\sum_{k \in M'_u} f_{ik} - ml_i)W_u, \qquad \forall i \in N', u \in U \tag{A.42}$$

The following block of constraints formulate bounds and integrality constraints on the variables.

$$left_{ij}, b_{ij}, ml_{ij} \in \{0, 1\}, \qquad \forall i, j \in N \cup N' \tag{A.43}$$

$$v_{ij} \in \{0, 1\}, \qquad \forall i < j, i, j \in N \cup N' \tag{A.44}$$

$$0 \leq x_i \leq W, \qquad \forall i \in N \cup N' \tag{A.45}$$

$$0 \leq y_i \leq L, \qquad \forall i \in N \cup N' \tag{A.46}$$

$$0 \leq c_i \leq C_{max}, \qquad \forall i \in N \cup N' \tag{A.47}$$

$$0 \leq C_k \leq C_{max}, \qquad \forall k \in M \cup M' \tag{A.48}$$

$$f_{ik} \in \{0, 1\}, \qquad \forall i \in N, k \in M \tag{A.49}$$

$$f_{ik} \in \{0, 1\}, \qquad \forall i \in N', k \in M' \tag{A.50}$$

$$z_k \in \{0, 1\}, \qquad \forall k \in M \cup M' \tag{A.51}$$

$$seq_k \in \{0, 1\}, \qquad \forall k \in M \cup M' \tag{A.52}$$

$$T_{max} \geq 0 \tag{A.53}$$

B

Exact lock scheduling run-time analysis

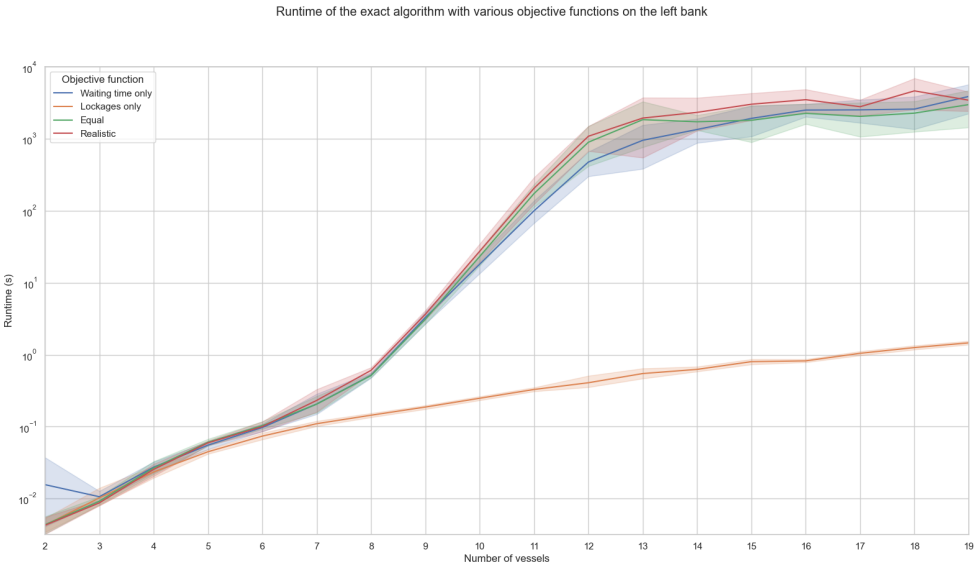


Figure B.1: Run-time of the exact algorithm for various objective functions on the left bank of the Port of Antwerp.

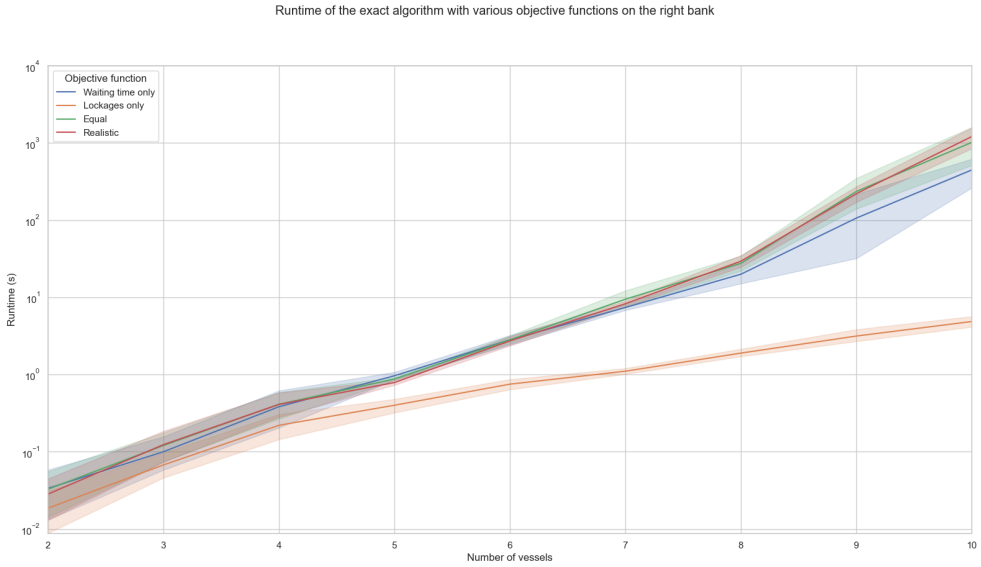


Figure B.2: Run-time of the exact algorithm for various objective functions on the right bank of the Port of Antwerp.

C

Paper

Algorithms for the online lock scheduling problem

Rico Hageman Neil Yorke-Smith

***Abstract*—We introduce the online variant of the Single Lock Scheduling Problem (SLSP-ONLINE). Three effective and deterministic algorithms based on an algorithmic framework are constructed and used to approach this problem. A benchmark with an offline exact algorithm shows that the algorithms are competitive in terms of solution quality on small problem instances. By assuming a constant lockage duration during scheduling, we aim to reduce the number of interactions with vessels. We demonstrate in simulations based on realistic arrivals that the abstraction reduces the interactions significantly with only a small effect on the tardiness of vessels.**

***Index Terms*—Lock scheduling problem**

I. Introduction

Maritime infrastructure projects are costly and have a long lifespan. It is, therefore, crucial to identify potential bottlenecks early and resolve them efficiently. By simulating the arrivals of vessels and their movements throughout the port realistically, this can be achieved. In addition, simulation can aid in determining the effectiveness of said projects to resolve the bottleneck.

The lock scheduling problem only recently gained traction compared to other parts of maritime and port related optimisation problems. The problems studied in the literature are often small with arrivals of a few days. In addition, the algorithms are designed to solve the offline variant of the problem where all the arrivals are known at the start and no restrictions are placed on the lock schedule by other schedules.

With this paper, we make the following contributions to the existing literature. First, we introduce the online variant of the single lock scheduling problem (SLSP-ONLINE) and argue when it is relevant to solve it.

Second, we introduce and benchmark three online scheduling algorithms to solve the posed problem using a constant lockage duration. They differ in their ability to affect already scheduled vessels. On small instances, they are competitive with exact offline methods.

Third, the trade-off between solution quality and the interactions with vessels due to a constant lockage duration during scheduling is discussed. We demonstrate on large realistic problem instances that the number of interactions can be significantly reduced in exchange for a small effect on the schedule quality.

II. Related work

The lock scheduling problem knows all kinds of variants and abstractions with their own complexity and solution methods. In general, without abstractions, the problem is NP-hard due to the hidden 2D bin packing problem when placing vessels inside a lockage.

When considering the single-chamber lock scheduling problem with a constant lockage duration and a capacity of a single vessel, a polynomial-time dynamic program exists [1]. Removing the capacity restrictions completely, allows one to create an algorithm based on the shortest path in an acyclic graph [2]. As long as the capacity of the chamber can be represented by an integer it is possible to minimise the total waiting time for the same problem with a single lock containing parallel chambers using a dynamic problem [3].

Complete and detailed lock schedules require a position for each vessel in their lockage. This vessel placement sub-problem is related to a well-known NP-complete problem, 2D bin-packing, with additional constraints. Verstichel and Vanden Berghe [4] composed an extended overview of all the different requirements including illustrations. Different mixed-integer linear programs are created to solve this lock scheduling problem. Formulations based on the job-shop scheduling problem are used for the single and sequential lock scheduling problems [5, 6]. In addition, a multi-commodity network formulation is used to solve the sequential lock scheduling problem [7]. Each of the mentioned exact algorithms has an unpredictable large run-time, which can reach over 16 hours for problems with 20 vessels and two chambers. In addition, they do not include the number of empty lockages in the objective function.

Verstichel et al. [8] introduced different methods for solving the vessel placement sub-problem. First, it presents an exact MILP which also formed the basis of several of the exponential algorithms presented earlier. Then two heuristics are presented which are not computationally intensive and produce solutions with a small optimality gap. These heuristics are used in combination with a benders decomposition to reduce the run-time [9].

Neighbourhood search is a commonly used meta-heuristic to solve the lock scheduling problem. Verstichel et al. [10] presented a neighbourhood search for the lock scheduling problem with a single lock with identical chambers where the solution is defined by an ordering of vessels. Provided with this ordering, heuristics will determine the feasibility of the vessel placement sub-problem and the start time is defined by the latest arriving vessel. Verstichel and Vanden Berghe [11] explored the same problem and approach but applied late acceptance criteria. Prandstetter et al. [12] solves the inter-dependent lock scheduling problem with identical chambers. It presents operations that destroy and repair a schedule in one pass and are specifically crafted for the problem at hand. Ji et al. [13] applies an adaptive large neighbourhood search based on the vehicle routing problem and uses general destroy

and repair operations to solve the generalised lock scheduling problem.

Summarising, the literature does not explain how to solve a lock scheduling problem relevant for simulations as the available solution methods fail to simultaneously address all of the following (1) sub-second run-time per vessel, (2) realistic objective function and (3) allows vessels to arrive in an online fashion.

III. Problem statement

A. Problem motivation

Simulations of a port consist of vessels that traverse several restricting elements like tide windows and locks. In addition, the arrival of a vessel at its destination is scheduled to prevent multiple vessels moor to the same spot. Finding a feasible solution for this problem is hard given that scheduling the sub-components in isolation is often NP-hard. It is therefore not feasible to find a global solution in one pass. Using a backtracking algorithm it is possible to find a feasible route for each vessel. For such an approach to work, it is required that the schedules of the restricting elements can be adapted to every occurring event.

For a lockage schedule, this implies, that vessels can be added to existing lockages and that the start time and duration of lockages can be changed. Such changes require vessels in the lockage to adapt their existing journey and might need to be rescheduled at other restricting elements. Cascading changes like these need to be handled by the backtracking algorithm and are deemed out of scope for this work. However, an idea to limit the number of times cascading changes could potentially be required is explored.

Depending on the layout of the port, a vessel can traverse multiple locks during its journey. However, the vessel might encounter other restricting elements in between the locks. It is up to the backtracking algorithm to align each of these passages. Therefore, the lock scheduling problem examined in this work consists of a single lock with multiple heterogeneous chambers.

B. Formal problem formulation

In the posed online lock scheduling problem there is a single lock with m heterogeneous chambers and a sequence of n vessels that arrive online. Let $C = \{c_1, ..., c_m\}$ be the set of chambers. Then, let L_c be the length, W_c the width and P_c the processing duration of chamber $c \in C$.

Let $V = \{v_1, ..., v_n\}$ be the set of vessels. With l_v the length, w_v the width and p_v the entry and exit duration of vessel $v \in V$. In addition, the arrival of vessel v is defined by its direction d_v and the arrival time a_v. There are two types of directions, either up- or downstream.

Let $L_c = \{l_1, ..., l_k\}$ be the set of lockages processed by chamber $c \in C$ and $L = \bigcup_{c=1}^{m} L_c$ be the set of all lockages. Then, let d_l be the direction of lockage $l \in L$ and s_l and e_l its start and end times.

Each chamber, c, processes its set of lockages, L_c, in alternating direction. Consecutive lockages l and l' must adhere to the following set of constraints where B is a constant buffer between sequential lockages;

$$s_{l'} \geq e_l + B \tag{1}$$

$$d_l \neq d_{l'} \tag{2}$$

Let $V_l = \{v_1, ..., v_n\}$ be the set of vessels processed by lockage l. A vessel can only be processed by a lockage if the lockage is in the same direction as the vessel arrives. Equations (3) and (4) ensure that every vessel is processed by a single lockage. Equation (5) enforces that the direction of a vessel and the lockage it is placed in are in the same direction.

$$V_l \cap V_{l'} = \emptyset \qquad \forall l, l' \in L \tag{3}$$

$$v \in V_l \qquad \exists l \in L, \forall v \in V \tag{4}$$

$$d_v = d_l \qquad \forall v \in V_l, l \in L \tag{5}$$

The duration of a lockage is dependent on its chamber and the vessels it processed. In addition, when vessels arrive not exactly after each other, the duration increases. Each vessel retrieves a requested time of arrival (ra_v). Equation (6) ensures that the requested time of arrival is after the first moment the vessel actually can arrive. Equation (7) ensures enough time between the sequential arrival of the vessels v and v'.

$$ra_v \geq a_v \qquad \forall v \in V \tag{6}$$

$$ra_{v'} \geq ra_v + p_v \tag{7}$$

The start and end time of a lockage are defined by Equations (8) and (9). Finally, Equation (10) is an helper variable that is used in the objective function.

$$s_l = \min_{v \in V_l}(ra_v) \qquad \forall l \in L \tag{8}$$

$$e_l = \max_{v \in V_l}(ra_v + p_v) + P_c + \sum_{v \in V_l} p_v \qquad \forall l \in L_c, c \in C \tag{9}$$

$$e_v = e_l \qquad \forall v \in V_l, l \in L \tag{10}$$

A schedule requires to have a fixed position for each vessel in a lockage. This position is defined by a x_v and y_v coordinate relative to the bottom left of the chamber. Equations (11) and (12) ensure that a vessel is placed within the dimensions of the chamber. A vessel must be moored onto the left or the right side of the chamber. The binary decision variable ml_v indicates if vessel v is moored onto the left side. Equation (13) enforces the correct x coordinate of vessel v based on the side it is moored on.

$$0 \leq x_v \leq W_c - w_v \qquad \forall v \in V_l, l \in L_c, c \in C \tag{11}$$

$$0 \leq y_v \leq L_c - l_v \qquad \forall v \in V_l, l \in L_c, c \in C \tag{12}$$

$$x_v = \begin{cases} 0 & \text{if } ml_v = 1, \\ W_c - w_v & \text{if } ml_v = 0 \end{cases} \qquad \forall v \in V_l, l \in L_c, c \in C \tag{13}$$

When multiple vessels are positioned within a lockage, they are not allowed to overlap. The functions $same\_side(v, l)$ returns the vessels in the same lockage l on the same side of vessel v. Between vessels $v, v' \in V_l$, it is required to have a minimum vertical distance of $vdistance(v, v')$ when moored onto the same side of the chamber. Equations (14) and (15) ensure that two vessels on the same side do not overlap and that the minimum distance is enforced.

$$|y_v - y_{v'} - l_{v'}| \geq vdistance(v, v') \quad \forall v' \in same\_side(v, l), v \in V_l, l \in L \quad (14)$$

$$y_v \leq y_{v'} + l_{v'} \vee y_v \geq y_{v'} + l_{v'} \quad \forall v' \in same\_side(v, l), v \in V_l, l \in L \quad (15)$$

When vessels share the same vertical position on opposite sides of the chamber, a minimum horizontal distance between them is required. The function $opposite\_side\_overlapping(v, l)$ returns the set of vessels in l that are moored on the opposite side of vessel v and share some overlapping y position. The horizontal distance required between these vessels are retrieved with $hdistance(v, v')$. Equation (16) enforces the additional horizontal distance for these vessels.

$$|x_v + l_v - x_{v'}| \geq hdistance(v, v') \quad \forall v' \in opposite\_side\_overlapping(v, l), v \in V_l, l \in L \quad (16)$$

Equations (17) to (25) summarise the different decision variables and list their domain. The requested arrival times of a vessel at a lock is defined up to a minute. This can be modelled as an integer since the first event in the problem. It is therefore also required to provide the parameters as an integer representing the minutes since the same moment.

$$d_v, v \in V, d_v \in \{0, 1\} \quad (17)$$
$$x_v, v \in V, x_v \in \mathbb{Z}, 0 \leq x_v \leq max_{c \in C}(W_c) \quad (18)$$
$$y_v, v \in V, y_v \in \mathbb{Z}, 0 \leq y_v \leq max_{c \in C}(L_c) \quad (19)$$
$$ra_v, v \in V, ra_v \in \mathbb{Z}, 0 \leq ra_v \quad (20)$$
$$e_v, v \in V, e_v \in \mathbb{Z}, 0 \leq e_v \quad (21)$$
$$V_l, l \in L, V_l \subseteq V \quad (22)$$
$$d_l, l \in L, d_l \in \{0, 1\} \quad (23)$$
$$s_l, l \in L, s_l \in \mathbb{Z}, 0 \leq s_l \quad (24)$$
$$e_l, l \in L, e_l \in \mathbb{Z}, 0 \leq e_l \quad (25)$$

C. Objective function

The goal of a lock scheduling algorithm is to find a lock schedule where both the delay for all the vessels and the number of lockages used are minimised. In addition, outliers, where vessels have extreme delays, are to be avoided. This can be achieved by including the maximum delay or squaring individual delays.

The delay of a vessel can be measured using either waiting time or tardiness. This is defined as the difference between the arrival time of a vessel and the start and end time of a lockage respectively. In a detailed lock schedule, a lockage can start before the arrival of any vessel it processes. Therefore, tardiness is always used when detailed lock schedules are compared. Each experiment will define the objective function component weights (K_i).

1) **Squared waiting time:** $K_1 \sum_{v \in V}(ra_v - a_v)^2 + K_2 \sum_{l \in L}(1)$
2) **Squared tardiness:** $K_1 \sum_{v \in V}(e_v - a_v)^2 + K_2 \sum_{l \in L}(1)$
3) **Summed waiting time:** $K_1 \sum_{v \in V}(ra_v - a_v) + K_2 \max_{v \in V}(ra_v - a_v) + K_3 \sum_{l \in L}(1)$
4) **Summed tardiness:** $K_1 \sum_{v \in V}(e_v - a_v) + K_2 \max_{v \in V}(e_v - a_v) + K_3 \sum_{l \in L}(1)$

IV. Algorithms

This section introduces three different online algorithms for the posed lock scheduling problem. It does so, by first explaining a framework in which the algorithms have to operate. Then, the different algorithms are presented. All three approaches are inexact methods as the online nature of the decision does not find the global optimal schedule. The motivation for the straightforward algorithms is the fact that the problem is unexplored and a baseline is required.

A. Algorithmic framework

Information about a vessel comes available over time when a vessel arrives close to the Port. It is therefore important to be able to decide on a single vessel. Algorithm 1 presents pseudo-code representing the framework in which all the algorithms will operate. During the first step, the lock scheduling algorithms return a set of proposals to accommodate the vessel. Then, out of all the options, a single proposal is selected. This is based on the objective function and the feasibility of the proposal regarding other schedules at the port. Finally, the selected proposal has to be executed onto the lock schedule to finalise it.

Every proposal has to be evaluated before a selection can be made. It is trivial to determine the objective difference regarding the lock schedule. However, when the algorithms are used in a simulation of the complete port, this becomes more complex. It is therefore desirable to minimise the number of interactions with vessels. This is achieved by assuming a constant lockage duration during the scheduling phase. This allows the algorithms to add vessels to a lockage without the requirement of interacting with each vessel in that lockage to ensure the new duration is fine by them. During the experiments, a constant lockage duration of 60 minutes is assumed. This allows at least one vessel to be added to the lockage, have the lock move to the other direction and have some additional time left. The number of vessels in the lockage may require more time than the constant lockage duration. During the conversion, care will be taken to make room for such lockages.

B. Online lock scheduling algorithms

Three online scheduling algorithms are created which differ in their abilities to affect vessels and lockages different from

Algorithm 1 Framework for the online lock scheduling algorithms

Require: $vessel$, $arrivalTime$, $direction$
$allProposals \leftarrow$ DetermineAllProposals($vessel$, $arrivalTime$, $direction$)
$selectedProposal \leftarrow$SelectProposal($allProposals$)
ExecuteProposal($selectedProposal$)

the vessel currently scheduled. Algorithm 2 presents the logic which all the algorithms share. When a vessel cannot be allocated with the current arrival time, the algorithm will delay the vessel and try again until at least a single solution is found. For a lockage to accommodate a vessel, it must traverse in the right direction, the vessel must arrive before the start of a lockage and the vessel placement sub-problem must be solved for the vessels already in the lockage together with the currently scheduled vessel.

The difference between the online scheduling algorithms are implemented in the feasibility checks and explained as follows;

1) **Default:** This algorithm is not able to delay other lockages. A new lockage is therefore only created if there is enough time between the predecessor and the successor. As the lockage duration is assumed to be constant a vessel can always be added to a lockage if the vessel placement sub-problem can be solved.
2) **Non-Greedy:** This algorithm extends the default algorithm and is allowed to delay other lockages. Therefore it will always propose solutions for creating a lockage as it will push its successors onto later in time.
3) **Improved:** An extension of the Non-Greedy algorithm where the proposals of the Non-Greedy algorithm are examined and improved by local search.

C. Local search

The Improved lock scheduler is equipped with capabilities to revisit made decisions to further improve the quality of the resulting lock schedules. These capabilities are based on the fact that sub-optimal decisions are made, especially when vessels are planned in non-chronological ordering. Figure 1 is an example where the ordering in which the vessels are added by the algorithm affects the outcome significantly when all the vessels could arrive at the same moment. The proposals of the Improved lock scheduler found using the local search operators are always in addition to the existing proposals.

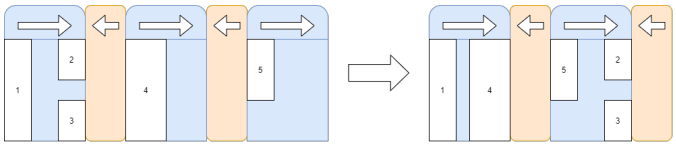


Fig. 1: Example where the ordering in which vessels are added affect the number of lockages required to allocate all vessels.

The first local search operator examines the first two lockages in the same direction either before or after the moment a new lockage is created. When different subsets of the current lockages and the new vessel could fit in only two lockages, it is proposed to omit the additional lockage and reassign the vessels to their new lockage.

The second local search operator compares only two lockages. The lockage with the currently added vessel and the first predecessor in the same direction. When a vessel in the predecessor is delayed less compared to the new vessel, it is proposed to swap them. This is aimed to reduce the maximum waiting time, as the total delay remains the same.

D. Converting abstract lock schedules

When scheduling with a constant lockage duration, the algorithms often create lockages that will take more time in reality. Therefore, an abstract schedule must be converted into a detailed schedule. A simplistic approach, which does no additional optimisation, is the following push-back method.

1) Create an empty detailed lock schedule.
2) Starting at the first lockage in the abstract schedule, the actual lockage duration is determined based on the vessels allocated.
3) Based on the predecessor in the detailed lock schedule and the vessels inside the lockage, the earliest start-time of the lockage is determined.
4) A new lockage is added to the detailed lock schedule with the updated start time and duration.
5) The conversion continues with the next lockage in the abstract schedule at step 3.

Note that using this method, lockage gets pushed back towards a later start time if a lockage turns out to use more than the time anticipated. However, the contrary is also true; when a lockage takes less time than scheduled, the sequential lockage could start earlier if the vessels can arrive at that moment.

V. Experimental setup

A. Datasets

The Port of Antwerp provided all the lock movements of the year 2019. This dataset forms the basis for the experiments in this work. The port utilises locks to separate two banks from the tide of the river the Westerschelde. Each bank has multiple entry points at different points along the river. As the algorithms introduced in this work are made for a single lock with multiple chambers, each bank is reduced to its main entry point during experiments. By redirecting the traffic of the other locks towards these main locks, the demand will become larger than in reality. This is by no means a simplification of the problem.

Provided with this set of vessels that need to be processed by a lock, it is important to determine a realistic moment for

Algorithm 2 Basic algorithm for the online lock scheduling algorithms.

Require: $lockSchedule$, $vessel$, $arrivalTime$, $direction$
  if $lockSchedule$ is empty **then**
    Propose to create lockage at $arrivalTime$
    Return
  end if

  $lockageBeforeVesselArrival \leftarrow DetermineLockageBeforeVesselArrival$(arrivalTime)
  $lockageAfterVesselArrival \leftarrow DetermineLockageAfterVesselArrival$(arrivalTime)

  if DoesLockageAccomodateVessel($lockageBeforeVesselArrival$, $vessel$) **then**
    Propose to add vessel to $lockageBeforeVesselArrival$
  end if

  if DoesLockageAccomodateVessel($lockageAfterVesselArrival$, $vessel$) **then**
    Propose to add vessel to $lockageAfterVesselArrival$
  end if

  if CanCreateLockageAfter($lockageBeforeVesselArrival$, $vessel$) **then**
    Propose to create lockage after $lockageBeforeVesselArrival$
  end if

  if CanCreateLockageAfter($lockageAfterVesselArrival$, $vessel$) **then**
    Propose to create lockage after $lockageAfterVesselArrival$
  end if

  if no proposal is created **then**
    Delay vessel until end of $lockageAfterVesselArrival$ and reschedule it
  end if

which the vessel requests to be scheduled in the lock. When using the realised arrival times as input for the lock scheduling algorithms it becomes trivial to generate a schedule. Therefore, based on the hourly, daily and monthly arrival patterns in the dataset and the number of vessels new arrivals are generated. These arrivals are based on their arrival at the start of the Westerschelde. This will spread out the vessels as they do not travel the river at an equal speed and requires the algorithms to create a schedule by themselves.

TABLE I: Characteristics of the main entrance locks at the Port of Antwerp.

Lock	Length (m)	Width (m)	Main entrance to
Zandvlietsluis	500	57	Right bank
Berendrechtsluis	500	68	Right bank
Kieldrechtsluis	500	66	Left bank

B. Run-time analysis

The run-time of the algorithms is dependent on two factors; the inter-arrival time, which determines how close the vessels arrive at each other, and the planning horizon. Control over these parameters is gained by generating artificial problem instances. Arrival times are sampled from a Poisson process and the dimensions of each vessel are uniformly selected from all the arrivals of the 2019 dataset. The values for the parameters during the experiments are presented in Table II and are based on the data analysis on historical lockages. Arrivals are generated for both a single and double chamber lock. The dimensions are similar to those of the main entry points of the left and right bank respectively.

With these parameters, 25 instances will be generated for each parameter combination. Because each instance is a realisation of a random process the number of vessels per instance is not fixed. Therefore the run-time divided by the number of vessels is reported. The algorithms will solve every instance in both a chronological ordering and with 10 random arrival orders. All experiments are run on windows 10 with an Intel i7 processor and 12GB of RAM and are implemented in C#.

The framework for the online lock scheduling algorithms requires an objective function to select the best proposal. During these experiments, this objective is based on the summed waiting time objective function. Exact weights for each of these components are irrelevant for the run-time. No significant differences have been observed when the experiment is repeated with different objective functions and weights.

TABLE II: Parameters with their default value and range for the run-time analysis of the online lock scheduling algorithms.

Parameter	Unit	Default value	Range
Inter arrival time	hours	0.6	[0.3, 0,35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
Horizon	days	2	[1, 2, 4, 8, 16, 32, 64, 128, 256]

C. Comparison with exact algorithm

The lack of existing online lock scheduling algorithms requires to use of another baseline. By comparing with an exact offline algorithm, it is possible to determine the optimality gap. However, the comparison is unfair as the offline algorithm knows of every arrival from the beginning.

The mixed-integer linear program introduced by Verstichel et al. [5] is used as this baseline. Adaptions to the formulation regarding the representation of chambers as presented by Ji et al. [14] are applied. In addition, some constraints are removed as they do not apply to the lock scheduling problem of interest. Finally, the constraints for the vessel placement sub-problem are simplified as sea-going vessels cannot moor onto each other. Appendix A lists the complete model.

The literature review showed that none of the objective functions used for exact algorithms contains the number of empty lockages. Balakrishnan et al [15] describes how the formulation could be adapted to force an ordering in the decision variables seq_{kl} such that it helps to determine the number of empty lockages. However, it would also require significantly more variables and constraints to be added. To prevent increasing the complexity of the model further, it is opted to omit counting the empty lockages during this experiment.

During the following experiments, only the summed waiting time objective function is used. The squared waiting time objective function cannot be used as it would destroy the linear properties of the formulation. The different weights for the objective function components are listed in Table III.

From the historical arrivals, a continuously increasing sequence of arrivals is randomly selected until solving the problems becomes infeasible within a practical time limit of 12 hours. For each problem size, 10 instances are created for the exact algorithm. From these instances, 10 additional instances are created for the online lock scheduling algorithms by shuffling the arrival order randomly. Experiments are executed using Ubuntu 20.04.2 LTS, a dual-core processor and 32GB of RAM. The model is implemented in C# with the state-of-the-art CP-SAT hybrid solver from Google OR-Tools [16].

D. Relative comparison

Exact algorithms are not able to solve large problem instances within a reasonable amount of time. The online lock scheduling algorithms are therefore compared relative to each other on generated problem instances with a length of a year. The experiments are based on 15 sets of different arrivals and departures based on the historical patterns for both the left and right banks. Each set is solved ten times using a random arrival ordering and once with a chronological ordering.

The summed waiting time and squared waiting time objective functions are used during these experiments. For the squared objective function, the weight of the number of lockages is increased. The summed waiting time objective function has two experiments. First, the weight of the maximum waiting time is increased while the weight of the total waiting time and the number of lockages is set to 1 and 30 respectively. Then the weight of the lockages is increased with the weight of the total waiting time set at 1 and the maximum waiting time at 2. Both these objectives are similar to the realistic objective function as introduced during the experiment with the exact algorithm. However, this time the empty lockages are also included.

E. Benefit of constant lockage duration

The trade-off between interactions with vessels and the lock schedule quality is made by comparing lock schedules created using a constant lockage duration which are converted using the push-back method and lock schedule created with a realistic constant duration. During each step of the process, the number of times a vessel is asked to arrive at a certain moment is counted. The detailed lock schedule is created with the same logic of the Improved lock scheduling algorithm. However, this time the lockage duration is not fixed but based on the number of vessels in the lockage. Each of the locks has a processing duration of at least 30 minutes. In addition, 20 minutes are added for every vessel entering and exiting the lock.

Problem instances with a length of three months are generated 25 times based on the historical patterns for both the left and right banks. The problems are solved in both a random and chronological ordering. The abstract lock scheduling algorithms use a varying constant lockage duration to determine its effect.

For the summed tardiness waiting time objective, the weights of the realistic and waiting time only objective function are used as introduced in the experiment where the algorithms are compared to the exact algorithm. In addition, the squared tardiness waiting time is used with a fixed weight for the number of lockages of 900. Depending on the number of vessels, this is roughly similar to 30 minutes additional delay.

VI. Results

A. Run-time analysis

Figures 2 and 3 show the run-time of the algorithms on the generated instances on the instances with both a single and a double chamber lock. The trends in the results are similar for both graphs. As the arrivals are based on a random process, the

TABLE III: Overview of the different objective weights used in the exact lock scheduling experiments.

Identifier	Total waiting time	Maximum waiting time	Number of lockages
Equal objective	1	1	1
Waiting time only	1	1	0
Realistic objective	1	2	30
Lockage only	0	0	1

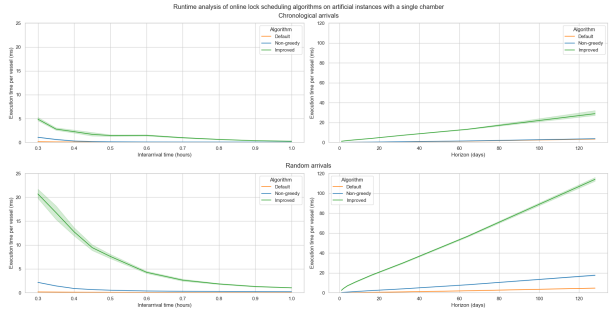


Fig. 2: Run-time of the online lock scheduling algorithms on artificial instances with a single chamber. Default values for the inter-arrival time and the horizon are 0.6 hours and 2 days respectively.

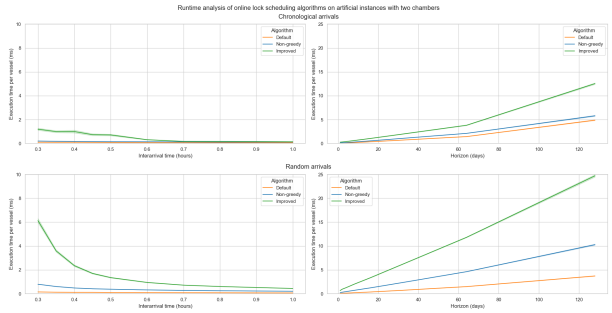


Fig. 3: Run-time of the online lock scheduling algorithm on artificial instances with two chambers. Default values for the inter-arrival time and the horizon are 0.6 hours and 2 days respectively.

number of vessels per instance differ. Therefore, the run-time is reported per vessel.

Decreasing the inter-arrival time increases the run-time of the algorithms. This is especially noticeable for the improved algorithm. Increasing the horizon of the experiment increases the run-time linearly for all algorithms. In addition, it is clear that chronological arrivals are beneficial in terms of run-time in comparison with random arrival orderings.

The linear increase in the run-time per vessel with a growing planning horizon is expected due to the data structures used to store the lock schedule. Decreasing the inter-arrival time increases the number of vessels per time interval. This allows more vessels to be allocated together in a lockage. Therefore, the Improved algorithm has more subsets of vessels to analyse. This difference is also present when the arrivals are not chronological ordered.

B. Comparison with exact algorithm

The relative performance of the online lock scheduling algorithm compared to the exact algorithm on problems of the left bank are depicted in Figures 4 and 5. As the online algorithms did not find different solutions for the equal, realistic and waiting time only objective functions, only the former is included. The same applies for the experiments with the right bank as presented by Figures 6 and 7.

The Non-Greedy and Improved online lock scheduling algorithms are competitive and outperform the Default algorithm consistently. However, there are only a few cases where the Improved algorithm is better than the Non-Greedy algorithm. Both often result in a lock schedule with fewer lockages while the waiting time is higher. Comparing the algorithms on the problem with a lockage only objective shows that the non-greedy and improved algorithms are good at minimising the number of lockages.

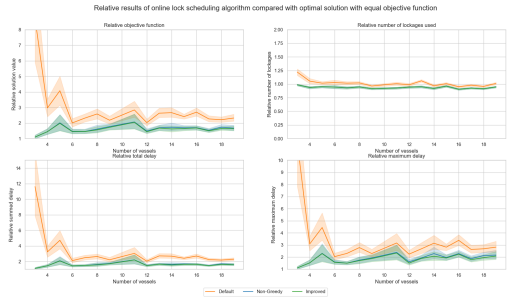


Fig. 4: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the left bank. The objective function is to minimise the total waiting time, the maximum waiting time and the number of lockages with equal waits.

C. Relative comparison

The results of the experiments with the realistic objective function are depicted in Figures 8 to 11. Increasing the weight of the maximum waiting time causes a small increase in the number of lockages. However, it creates a lock schedule where the maximum waiting time is drastically reduced while the average waiting time remains relatively constant. This trend applies to both banks and the Non-Greedy and Improved algorithms. The Default algorithm results in schedules with extreme maximum waiting times and is not shown in the graph.

Increasing the weight of the number of lockages with the same objective function causes both the average and maximum

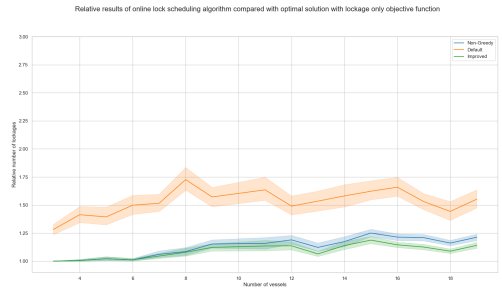


Fig. 5: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the left bank. The objective function is to minimise the number of lockages.

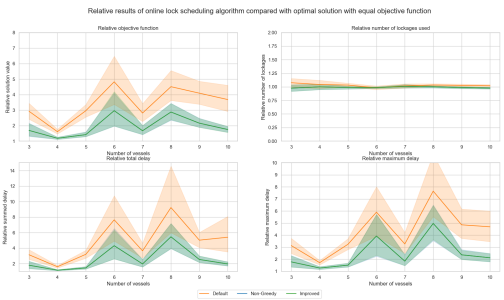


Fig. 6: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the right bank. The objective function is to minimise the total waiting time, the maximum waiting time and the number of lockages with equal waits.

waiting time to increase. When the number of lockages is drastically reduced compared to the schedule with a low weight, the average waiting time starts to rise quickly. Again, the Default algorithm is not competitive with the other two algorithms in terms of average and maximum waiting time.

Figures 12 and 13 shows the results of the squared objective function where the weight of the number of lockages is increased. The same trends as with the realistic objective function are present. However, this time the average waiting time increases harder and the maximum waiting time remains less compared to the other objective function. In addition, the weight of the number of lockages must be relatively high to reach a similar number of lockages as the realistic objective function.

For every objective function, it holds that the Non-Greedy and Improved algorithm have a lower average waiting time and user number of lockages when the ordering of vessels is random. However, the maximum waiting time is larger compared to the chronological ordering. These two algorithms always perform better than the Default algorithm. The Improved algorithm also outperforms the Non-Greedy algorithm. It is especially good at reducing the number of lockages.

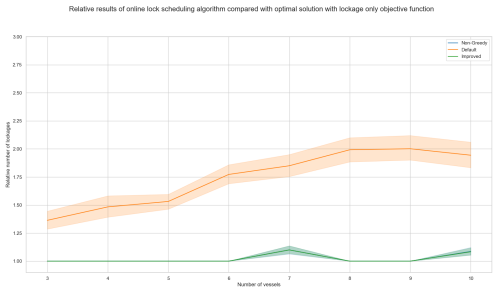


Fig. 7: Comparison of the online algorithms with the optimal solution on small realistic problem instances on the right bank. The objective function is to minimise the number of lockages.

D. Benefit of constant lockage duration

Figures 14 and 15 show the relative objective, its components and the number of interactions of the abstract algorithms compared to the detailed algorithm when using the realistic objective function. The improved abstract algorithm performs best and therefore its results are listed. The optimal relative objective is achieved with a constant lockage duration between 50 and 55 minutes. It reduces the interactions between 32 and 40% while increasing the average waiting time by 11 and 15%. Figures 16 and 17 and Figures 18 and 19 show the same results when using the squared and waiting time only objective functions respectively. Table IV summarises the results and shows the relative differences for the optimal constant lockage duration.

The trends are similar for every objective function. When the constant lockage duration increases the relative total and maximum tardiness increases. However, the number of lockages decreases. This is because when scheduling lockages with a long duration, more vessels can be added to it. Interestingly, even in the case that the number of lockages was not included, the abstract algorithms tend to use fewer lockages compared to the detailed algorithm.

E. Summary

The online lock scheduling algorithms can be compared relative to each other as following; The Default and Non-Greedy algorithms have the lowest run-time while the Improved algorithm produces the lock schedules of the highest quality. The latter is also confirmed when the algorithms are compared with the exact algorithm. On small problem instances, the Non-Greedy and Improved algorithms are competitive with each other.

Summarising the results presented related to the trade-off of using a constant lockage duration, Table IV shows the main result. The Improved lock scheduling algorithm reduces the number of interactions by more than 30% by using this abstraction. In addition, the number of lockages is less and the tardiness is slightly increased compared to a lock schedule created with a realistic lockage duration from the beginning.

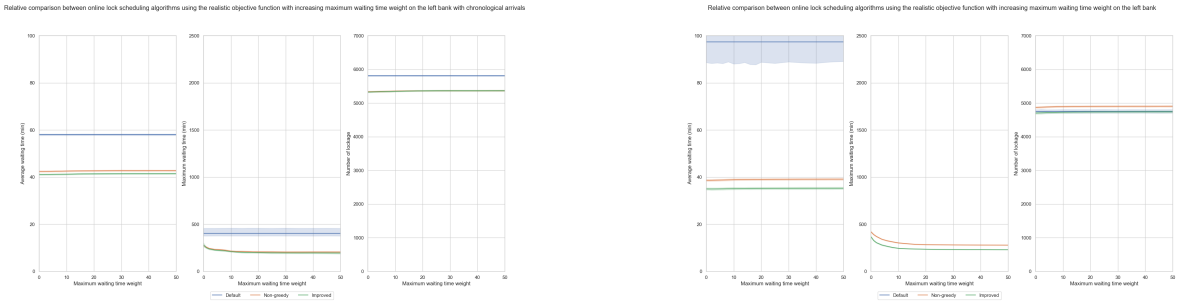
TABLE IV: Overview of the optimal constant lockage duration, the relative number of interactions and the different objective components for different objective functions with the improved abstract algorithm.

Objective function	Optimal constant lockage duration (min)	Relative objective function	Relative number of interactions	Relative average tardiness	Relative maximum tardiness	Relative number of lockages
Realistic	50-56	1.03-1.05	0.60-0.68	1.10-1.15	1.13-1.30	0.89-0.92
Squared	54-58	1.12-1.17	0.61-0.68	1.08-1.13	1.08-1.21	0.94-0.97
Waiting time only	44-48	1.05-1.07	0.58-0.65	0.98-1.02	1.00-1.15	0.93-0.97

VII. Conclusion and Future Work

This work introduced the online variant of the single lock scheduling problem called SLSP-ONLINE and argued its applicability for simulation purposes. We proposed three online algorithms which have a small run-time and can adapt existing lock schedules while taking into account a realistic objective function. By using a constant lockage duration we reduced the number of interactions between the lock scheduling algorithms and vessels significantly while the quality of the lock schedules is only slightly reduced.

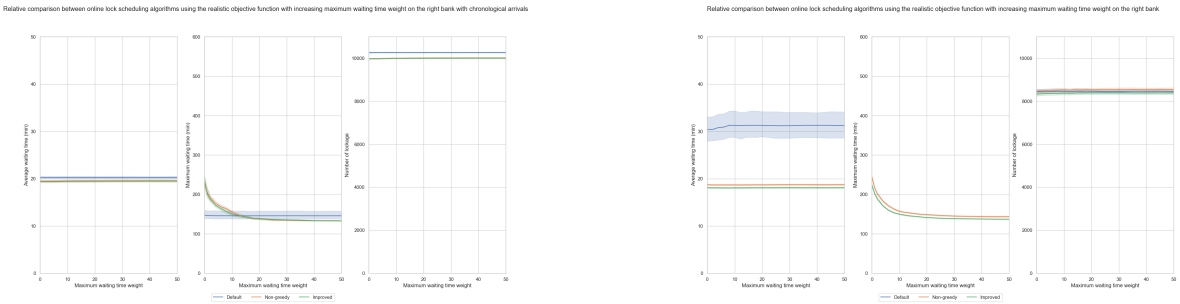
Based on the current work, there are different directions to take. First, additional local search operators could be added to the Improved lock scheduler and analysing the improvement of each operator allows making a better trade-off between additional run-time and lock schedule quality. Second, the concept of scheduling with a constant lockage duration and converting the schedule into a detailed lock schedule could be applied to the offline lock scheduling problem. Finally, when information about barges is available it is interesting to determine the performance of the algorithms again.



(a) Chronological ordering

(b) Random ordering

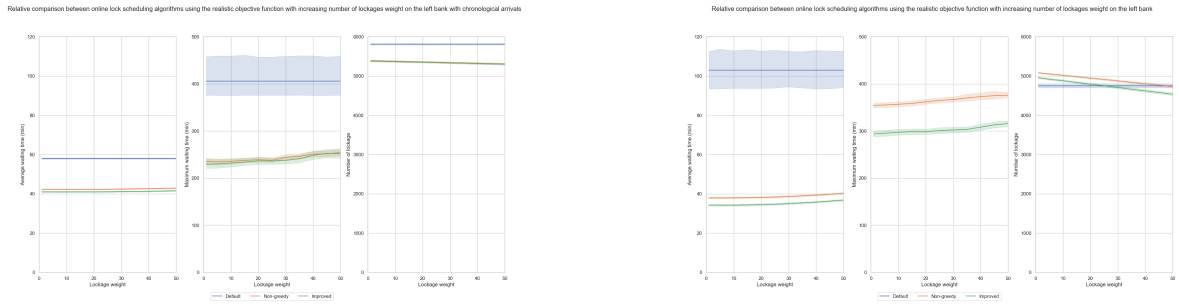
Fig. 8: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the left bank where the weight of the maximum waiting time is increased.



(a) Chronological ordering

(b) Random ordering

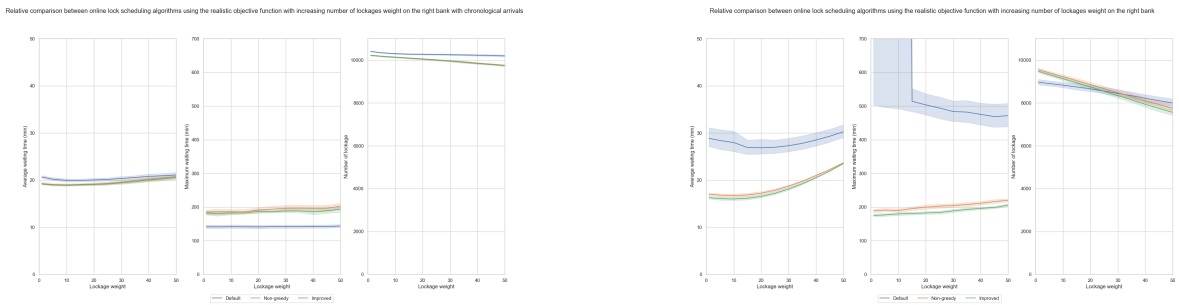
Fig. 9: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the right bank where the weight of the maximum waiting time is increased.



(a) Chronological ordering

(b) Random ordering

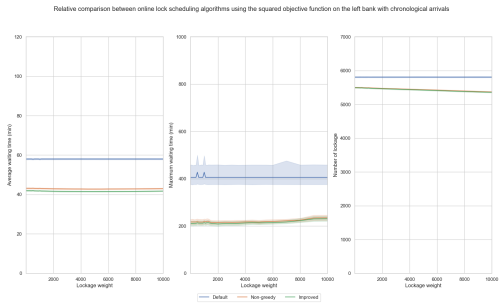
Fig. 10: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the left bank where the weight of the number of lockages is increased.



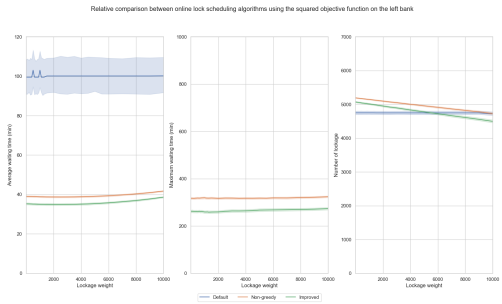
(a) Chronological ordering

(b) Random ordering

Fig. 11: Relative comparison of the online lock scheduling algorithms on realistic instances with the realistic objective function for the right bank where the weight of the number of lockages is increased.

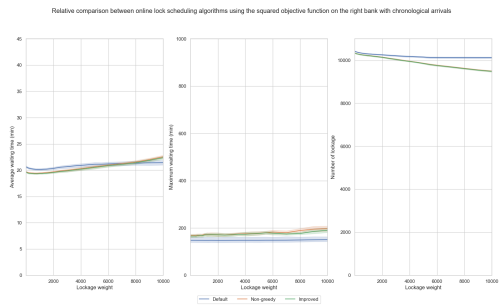


(a) Chronological ordering

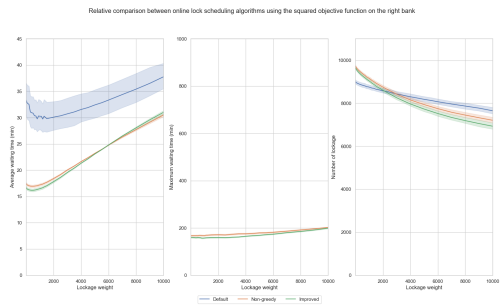


(b) Random ordering

Fig. 12: Relative comparison of the online lock scheduling algorithms on realistic instances with an objective function where individual waiting times are squared for the left bank.



(a) Chronological ordering



(b) Random ordering

Fig. 13: Relative comparison of the online lock scheduling algorithms on realistic instances with an objective function where individual waiting times are squared for the right bank.

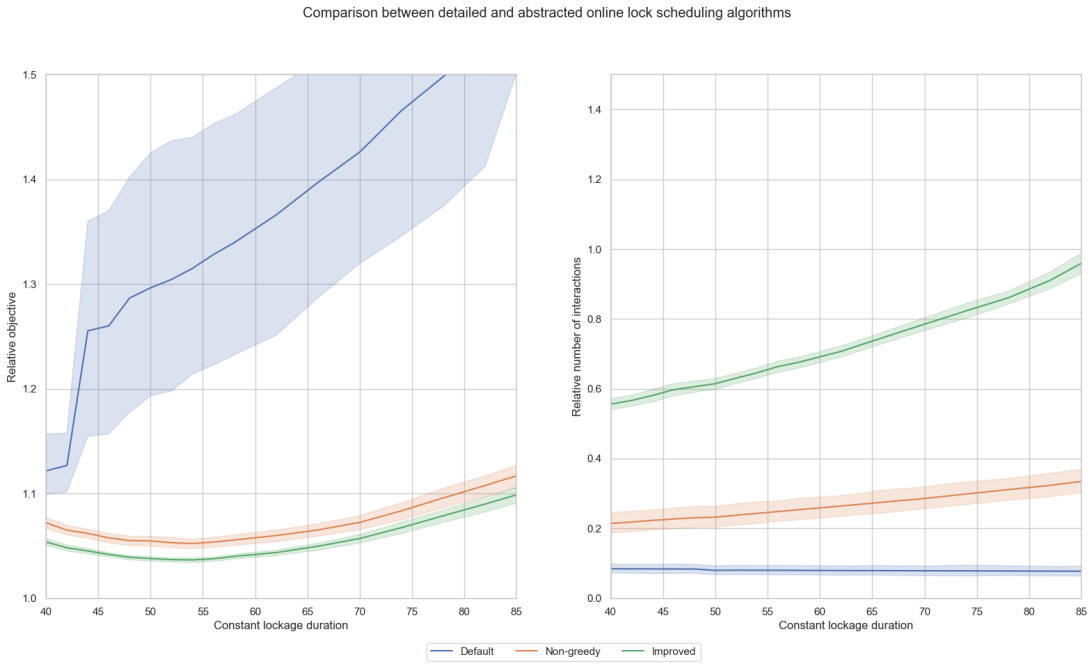


Fig. 14: Relative objective and fraction of interactions required for the abstract algorithms compared to the detailed online lock scheduling algorithm using the realistic objective function.

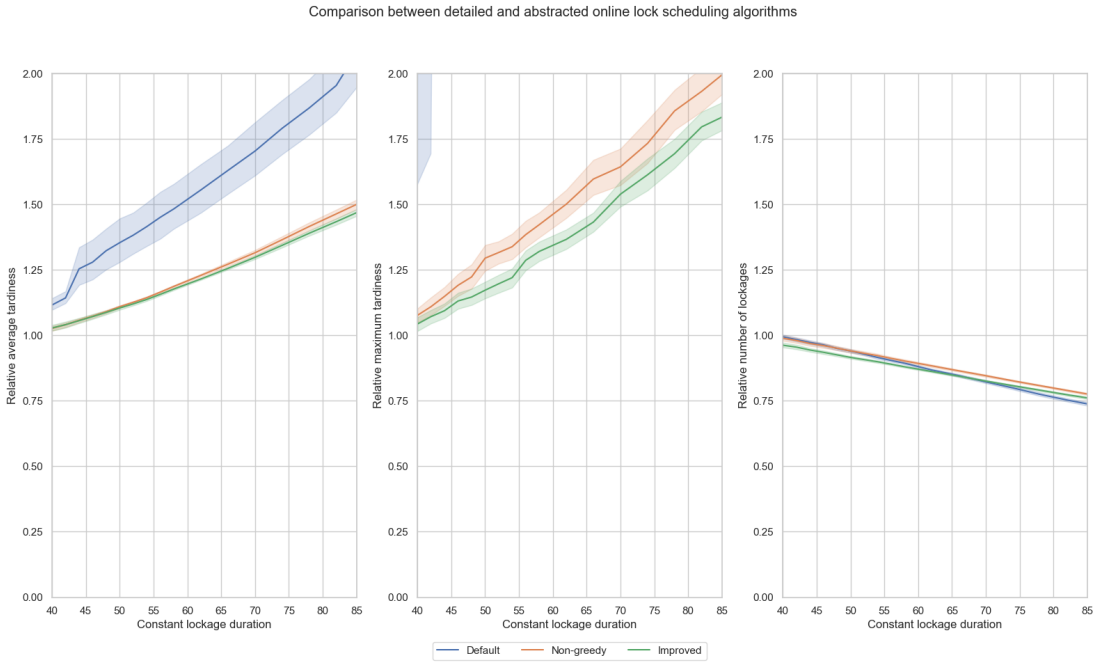


Fig. 15: Relative objective components of the resulting schedules of the abstract algorithms compared to the detailed online lock scheduling algorithm using the realistic objective function.

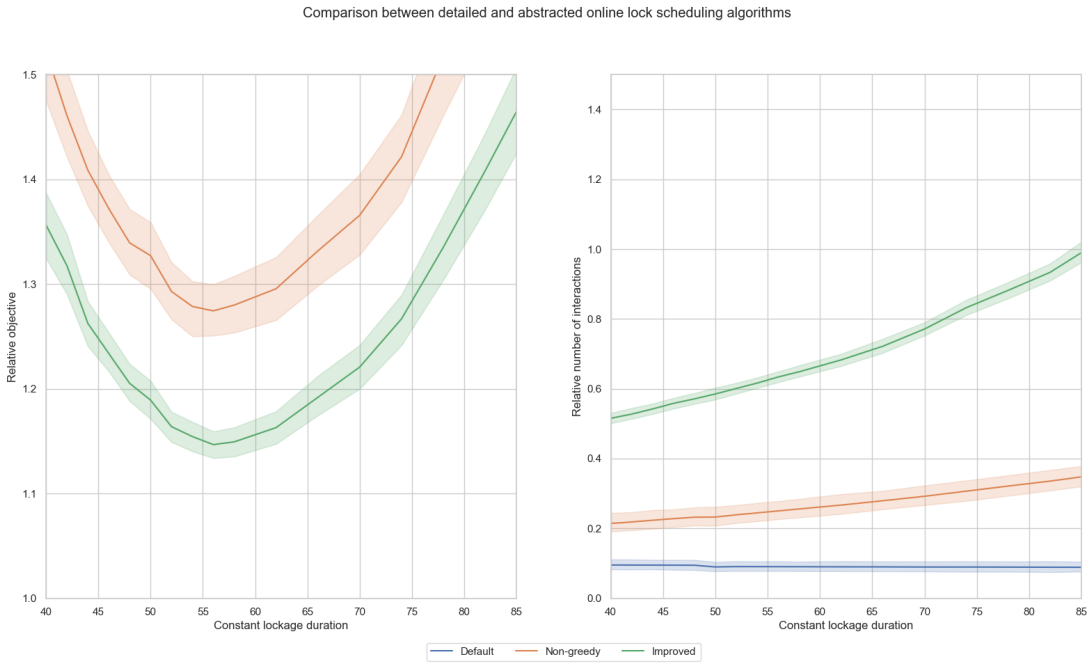


Fig. 16: Relative objective and fraction of interactions required for the abstract algorithms compared to the detailed online lock scheduling algorithm using the squared objective function.

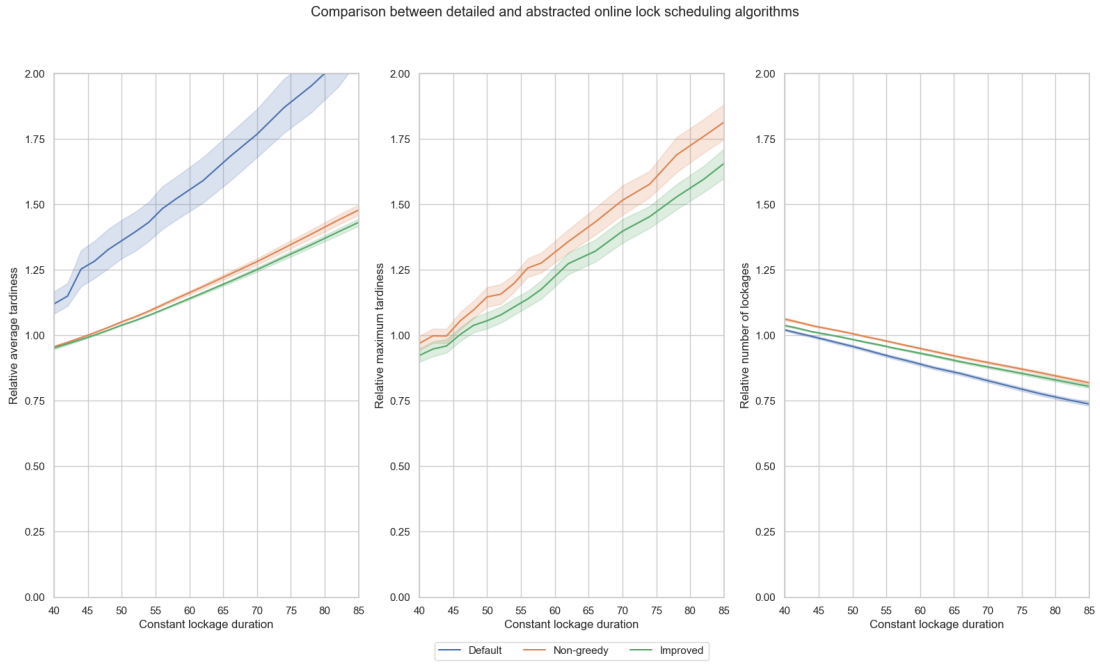


Fig. 17: Relative objective components of the resulting schedules of the abstract algorithms compared to the detailed online lock scheduling algorithm using the squared objective function.

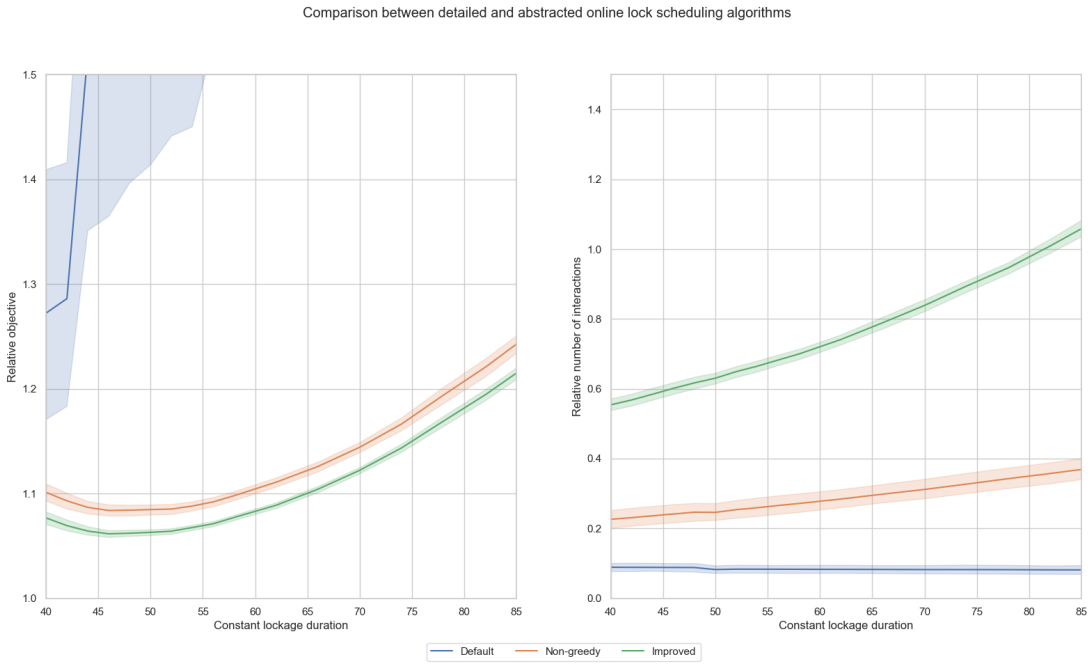


Fig. 18: Relative objective and fraction of interactions required for the abstract algorithms compared to the detailed online lock scheduling algorithm using the squared waiting time only function.

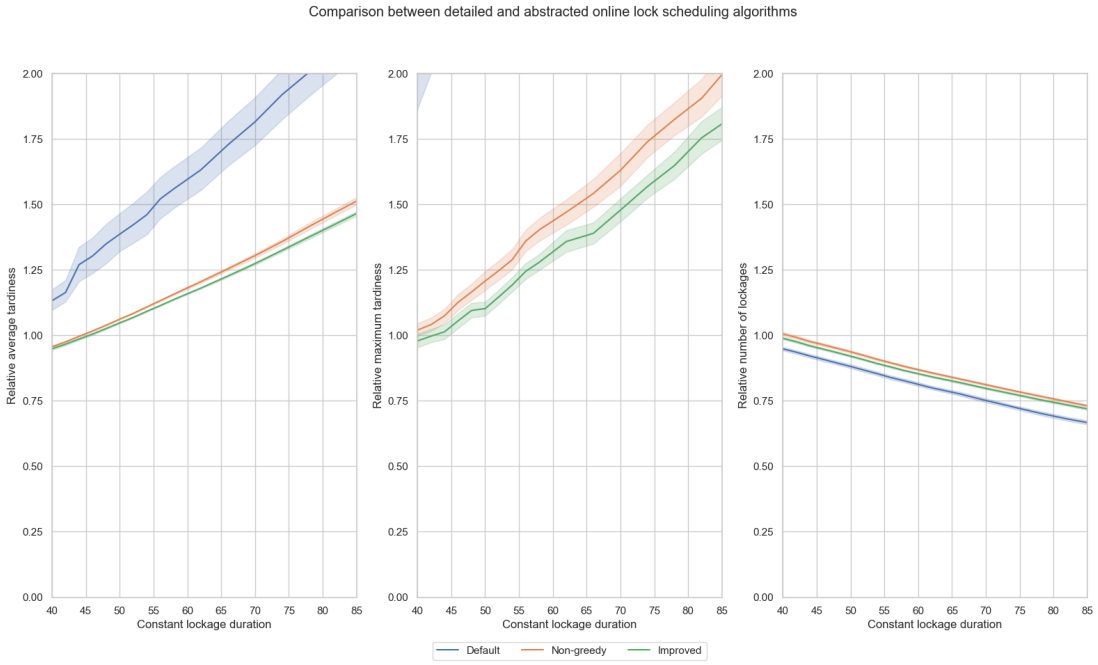


Fig. 19: Relative objective components of the resulting schedules of the abstract algorithms compared to the detailed online lock scheduling algorithm using the waiting time only objective function.

APPENDIX A
MIXED INTEGER LINEAR PROGRAM FORMULATION

A. Parameters

N, N':	Set of upstream, downstream vessels, indexed by i, j.
M, M':	Set of the upstream, downstream lockages, indexed by k, l.
w_i, l_i:	Width and length of upstream vessel i.
w_i', l_i':	Width and length of downstream vessel i.
dF_i, dB_i:	Minimal distance between upstream vessel i and the front, back of the chamber.
dF_i', dB_i':	Minimal distance between downstream vessel i and the front, back of the chamber.
sL_{ij}:	Minimal safety distance between upstream vessels i and j when they are lying behind each other.
sL_{ij}':	Minimal safety distance between downstream vessels i and j when they are lying next to each other.
sW_{ij}:	Minimal safety distance between upstream vessels i and j when they are lying behind each other.
sW_{ij}':	Minimal safety distance between downstream vessels i and j when they are lying next to each other.
r_i, r_i':	Arrival time of upstream, downstream vessel i at the coordination point.
U:	Set of physical chambers indexed by u.
W_u, L_u:	Width and length of the chamber u.
W, L:	Maximal width and length over all chambers
M_u, M_u':	Subset of M, reserved for upstream, downstream lockages performed on chamber u.
p:	Constant lockage duration.
$setup_{kl}$:	Minimal setup time between lockages k and l when they are processed by the same chamber. Depends on the direction of lockages k and l.
C_{max}:	Big M constant used as an upper bound for the completion time.

B. Variables

$\mathbf{x_i}$, $\mathbf{y_i}$:	Integer variables that define the x and y position of vessel i (front left corner).
$\mathbf{b_{ij}}$:	Binary variable indicating whether vessel $i$is to the left of vessel j or not.
$\mathbf{left_{ij}}$:	Binary variable indicating whether vessel i is behind vessel j or not.
$\mathbf{ml_i}$:	Binary variable indicating whether vessel i is moored onto the left side of the chamber or not.
$\mathbf{z_k}$:	Binary variable that indicates whether lockage k is used or not.
$\mathbf{f_{ik}}$:	Binary variable that indicates whether vessel i is processed in lockage k or not.
$\mathbf{v_{ij}}$:	Binary variable that indicates whether vessels i and j are processed in the same lockage or not.
$\mathbf{c_i}$:	Departure time of vessel i.
$\mathbf{C_k}$:	Completion time of lockage k.
$\mathbf{seq_{kl}}$:	Binary variable that indicated whether lockage k precedes lockage l or not.
$\mathbf{T_{max}}$:	Maximum waiting time of all vessels.

C. Objective function

The values of K_1, K_2 and K_3 are variable and are defined for each experiment performed with the algorithm.

$$K_1 \sum_{k \in N \cup N'} (c_i - p - r_i) + K_2 T_{max} + K_3 \sum_{k \in M \cup M'} Z_k \tag{26}$$

D. Constraints

The following blocks of constraints models the scheduling part of the lock scheduling problem.

$$c_i \geq C_{max}(f_{ik} - 1) + C_k, \qquad \forall i \in N, k \in M \tag{27}$$
$$c_i \leq C_{max}(1 - f_{ik}) + C_k, \qquad \forall i \in N, k \in M \tag{28}$$
$$c_i' \geq C_{max}(f_{ik} - 1) + C_k', \qquad \forall i \in N', k \in M' \tag{29}$$
$$c_i' \leq C_{max}(1 - f_{ik}) + C_k', \qquad \forall i \in N', k \in M' \tag{30}$$

$$p_k \geq p z_k, \qquad \forall k \in M \tag{31}$$
$$p_k \geq p z_k, \qquad \forall k \in M' \tag{32}$$

$$C_l - C_k \geq p_l + setup_{kl} - 2C_{max}(1 - seq_{kl}), \qquad \forall k < l \in M_u \cup M_u', u \in U \tag{33}$$
$$C_k - C_l \geq p_k + setup_{kl} - 2C_{max}(1 - seq_{kl}), \qquad \forall k < l \in M_u \cup M_u', u \in U \tag{34}$$

$$C_k - P_k \geq f_{ik} r_i, \qquad \forall i \in N, k \in M \tag{35}$$
$$C_k - P_k \geq f_{ik} r_i, \qquad \forall i \in N', k \in M' \tag{36}$$

$$Z_k \leq \sum_{i \in N} f_{ik}, \qquad \forall k \in M \tag{37}$$

$$Z_k \leq \sum_{i \in N'} f_{ik}, \qquad \forall k \in M' \tag{38}$$

The following block of constraints are transitive constraints and are used to break symmetry.

$$Z_{k+1} \leq Z_k, \qquad \forall k \in M_u \cup M'_u, u \in U \tag{39}$$

$$C_k \leq C_{k+1}, \qquad \forall k \in M_u \cup M'_u, u \in U \tag{40}$$

The following constraint is used for the objective function.

$$T_{max} \geq c_i - r_i - p, \qquad \forall i \in N \cup N' \tag{41}$$

The following blocks of constraints models the ship placement part of the lock scheduling problem for the upstream vessels.

$$left_{ij} + left_{ji} + b_{ij} + b_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geq 1, \qquad \forall i < j, i, j \in N, k \in M \tag{42}$$

$$x_i - x_j + W left_{ij} \leq W - w_i, \qquad \forall i, j \in N \tag{43}$$

$$y_i - y_j + L b_{ij} \leq L - l_i, \qquad \forall i, j \in N \tag{44}$$

$$x_j - x_i + (W + sW_{ij})(1 - left_{ij} + b_{ij}) \geq w_i + sW_{ij}, \qquad \forall i, j \in N \tag{45}$$

$$y_j - y_i + (L + sL_{ij})(1 - b_{ij} - left_{ij}) \geq l_i + sL_{ij}, \qquad \forall i, j \in N \tag{46}$$

$$x_i + w_i \leq W_u + (1 - f_{ik})W, \qquad \forall i \in N, k \in M_u, u \in U \tag{47}$$

$$y_i + l_i \leq L_u + (1 - f_{ik})L, \qquad \forall i \in N, k \in M_u, u \in U \tag{48}$$

$$y_i \geq dF_i, \qquad \forall i \in N \tag{49}$$

$$y_i + l_i \leq Lu - dB_i + (1 - f_{ik}), \qquad \forall i \in N, k \in M_u, u \in U \tag{50}$$

$$\sum_{k \in M} f_{ik} = 1, \qquad \forall i \in N \tag{51}$$

$$f_{ik} \leq Z_k, \qquad \forall i \in n, k \in M \tag{52}$$

$$x_i \leq (1 - ml_i)W, \qquad \forall i \in N \tag{53}$$

$$x_i + w_i \geq (\sum_{k \in M_u} f_{ik} - ml_i)W_u, \qquad \forall i \in N, u \in U \tag{54}$$

The following blocks of constraints models the ship placement part of the lock scheduling problem for the downstream vessels.

$$left_{ij} + left_{ji} + b_{ij} + b_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geq 1, \qquad \forall i < j, i, j \in N', k \in M' \tag{55}$$

$$x_i - x_j + W left_{ij} \leq W - w_i, \qquad \forall i, j \in N' \tag{56}$$

$$y_i - y_j + L b_{ij} \leq L - l_i, \qquad \forall i, j \in N' \tag{57}$$

$$x_j - x_i + (W + sW_{ij})(1 - left_{ij} + b_{ij}) \geq w_i + sW_{ij}, \qquad \forall i, j \in N' \tag{58}$$

$$y_j - y_i + (L + sL_{ij})(1 - b_{ij} - left_{ij}) \geq l_i + sL_{ij}, \qquad \forall i, j \in N' \tag{59}$$

$$x_i + w_i \leq W_u + (1 - f_{ik})W, \qquad \forall i \in N', k \in M'_u, u \in U \tag{60}$$

$$y_i + l_i \leq L_u + (1 - f_{ik})L, \qquad \forall i \in N', k \in M'_u, u \in U \tag{61}$$

$$y_i \geq dF_i, \qquad \forall i \in N \tag{62}$$

$$y_i + l_i \leq Lu - dB_i + (1 - f_{ik}), \qquad \forall i \in N', k \in M'_u, u \in U \tag{63}$$

$$\sum_{k \in M'} f_{ik} = 1, \qquad \forall i \in N' \tag{64}$$

$$f_{ik} \leq Z_k, \qquad \forall i \in n, k \in M' \tag{65}$$

$$x_i \leq (1 - ml_i)W, \qquad \forall i \in N' \tag{66}$$

$$x_i + w_i \geq (\sum_{k \in M'_u} f_{ik} - ml_i)W_u, \qquad \forall i \in N', u \in U \tag{67}$$

The following block of constraints formulate bounds and integrality constraints on the variables.

$$left_{ij}, b_{ij}, ml_{ij} \in \{0,1\}, \qquad \forall i,j \in N \cup N' \tag{68}$$

$$v_{ij} \in \{0,1\}, \qquad \forall i < j, i,j \in N \cup N' \tag{69}$$

$$0 \leq x_i \leq W, \qquad \forall i \in N \cup N' \tag{70}$$

$$0 \leq y_i \leq L, \qquad \forall i \in N \cup N' \tag{71}$$

$$0 \leq c_i \leq C_{max}, \qquad \forall i \in N \cup N' \tag{72}$$

$$0 \leq C_k \leq C_{max}, \qquad \forall k \in M \cup M' \tag{73}$$

$$f_{ik} \in \{0,1\}, \qquad \forall i \in N, k \in M \tag{74}$$

$$f_{ik} \in \{0,1\}, \qquad \forall i \in N', k \in M' \tag{75}$$

$$z_k \in \{0,1\}, \qquad \forall k \in M \cup M' \tag{76}$$

$$seq_k \in \{0,1\}, \qquad \forall k \in M \cup M' \tag{77}$$

$$T_{max} \geq 0 \tag{78}$$

References

[1] Jens Hermans. "Optimization of inland shipping". In: *Journal of Scheduling* 17.4 (2014), pp. 305–319.

[2] Ward Passchyn et al. "The lockmaster's problem". In: *European Journal of Operational Research* 251.2 (2016), pp. 432–441. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2015.12.007.

[3] Ward Passchyn, Dirk Briskorn, and Frits C. R. Spieksma. "No-Wait Scheduling for Locks". In: *INFORMS Journal on Computing* 31.3 (2019), pp. 413–428. DOI: 10.1287/ijoc.2018.0848.

[4] J. Verstichel and G.V. Berghe. *Scheduling serial locks: A green wave for waterbound logistics*. 2015, pp. 91–109. DOI: 10.1007/978-3-319-17419-8_5.

[5] Jannes Verstichel et al. "The generalized lock scheduling problem: An exact approach". In: *Transportation Research Part E: Logistics and Transportation Review* 65 (2014). Special Issue on: MODELING, OPTIMIZATION AND SIMULATION OF THE LOGISTICS SYSTEMS, pp. 16–34. ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2013.12.010.

[6] Bin Ji et al. "An exact approach to the generalized serial-lock scheduling problem from a flexible job-shop scheduling perspective". In: *Computers & Operations Research* 127 (2021), p. 105164. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2020.105164.

[7] Bin Ji et al. "Optimally solving the generalized serial-lock scheduling problem from a graph-theory-based multi-commodity network perspective". In: *European Journal of Operational Research* 288.1 (2021), pp. 47–62. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.05.035.

[8] J. Verstichel et al. "Exact and heuristic methods for placing ships in locks". In: *European Journal of Operational Research* 235.2 (2014). Maritime Logistics, pp. 387–398. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2013.06.045.

[9] J. Verstichel et al. "A Combinatorial Benders decomposition for the lock scheduling problem". In: *Computers & Operations Research* 54 (2015), pp. 117–128. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2014.09.007.

[10] Jannes Verstichel, Patrick De Causmaecker, and Greet Vanden Berghe. "Scheduling algorithms for the lock scheduling problem". In: *Procedia - Social and Behavioral Sciences* 20 (2011). 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, pp. 806–815. ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2011.08.089.

[11] Jannes Verstichel and Greet Vanden Berghe. "A Late Acceptance Algorithm for the Lock Scheduling Problem". In: Jan. 2009, pp. 457–478. ISBN: 978-3-7908-2361-5. DOI: 10.1007/978-3-7908-2362-2_23.

[12] Matthias Prandtstetter et al. "A Variable Neighborhood Search Approach for the Interdependent Lock Scheduling Problem". In: *Evolutionary Computation in Combinatorial Optimization*. Ed. by Gabriela Ochoa and Francisco Chicano. Cham: Springer International Publishing, 2015, pp. 36–47. ISBN: 978-3-319-16468-7.

[13] Bin Ji et al. "An adaptive large neighborhood search for solving generalized lock scheduling problem: Comparative study with exact methods". In: *IEEE Transactions on Intelligent Transportation Systems* 21.8 (2019), pp. 3344–3356.

[14] Bin Ji et al. "An exact approach to the generalized serial-lock scheduling problem from a flexible job-shop scheduling perspective". In: *Computers & Operations Research* 127 (Mar. 2021), p. 105164. DOI: 10.1016/j.cor.2020.105164.

[15] Nagraj Balakrishnan, John J. Kanet, and V. Sridharan. "Early/tardy scheduling with sequence dependent setups on uniform parallel machines". In: *Computers & Operations Research* 26.2 (1999), pp. 127–141. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(98)00051-3.

[16] Laurent Perron and Vincent Furnon. *OR-Tools*. Version 9.0. Google, Aug. 20, 2021. URL: https://developers.google.com/optimization/.