KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

The lockmaster's problem

Passchyn W, Coene S, Briskorn D, Hurink J, Spieksma F, Vanden Berghe G.

KBI_1530

The Lockmaster's Problem

Ward Passchyn[†] Sofie Coene[†] Dirk Briskorn[‡]
Johann L. Hurink[§] Frits C.R. Spieksma[†]
Greet Vanden Berghe[◊]

December 2015

Abstract

Inland waterways form a natural network infrastructure with capacity for more traffic. Transportation by ship is widely promoted as it is a reliable, efficient and environmental friendly way of transport. Nevertheless, locks managing the water level on waterways and within harbours sometimes constitute bottlenecks for transportation over water. The lockmaster's problem concerns the optimal strategy for operating such a lock. In the lockmaster's problem we are given a lock, a set of upstream-bound ships and another set of ships travelling in the opposite direction. We are given the arrival times of the ships and a constant lockage time; the goal is to minimize total waiting time of the ships. In this paper, a dynamic programming algorithm is proposed that solves the lockmaster's problem in polynomial time. This algorithm can also be used to solve a single batching machine scheduling problem more efficiently than the current algorithms from the literature do. We extend the algorithm so that it can be applied in realistic settings, taking into account capacity, ship-dependent handling times, weights and water usage. In addition, we compare the performance of this new exact algorithm with the performance of some (straightforward) heuristics in a computational study.

Keywords: *Transportation, Lock scheduling, Batch scheduling, Dynamic programming, Complexity*

[†]KU Leuven, Faculty of Economics and Business, ORSTAT, Naamsestraat 69, 3000 Leuven, Belgium

[‡]Chair of Production and Logistics, Bergische Universität Wuppertal, Rainer-Gruenter-Straße 21, 42119 Wuppertal, Germany

[§]University of Twente, Department of Applied Mathematics, P.O. Box 217, 7500 AE Enschede, The Netherlands

[◊]KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC, Gebroeders De Smetstraat 1, 9000 Gent, Belgium

e-mail addresses: {ward.passchyn, sofie.coene, frits.spieksma}@kuleuven.be; briskorn@uni-wuppertal.de; j.l.hurink@utwente.nl; greet.vandenberghe@cs.kuleuven.be

1 Introduction

Transportation of goods by ship, over sea as well as over waterways, is a promising alternative for transport over land. Reasons are its reliability, its efficiency (a barge of 3000 tons can transport as much as 100 train wagons or 150 trucks), its relatively low operating cost and its environmental friendliness. Hence, the relative importance of this mode of transport is rising. Maritime transport for intercontinental trade receives increasing attention due to the high cargo volumes that can be shipped at a much lower energy cost compared to air cargo traffic. Inland waterways transport is therefore seen as a mode of transport that can make a significant contribution to sustainable mobility. For instance, in a recent report, the European Commission (2015) promotes a better use of inland waterways in order to relieve heavily congested transport corridors. Not only is the energy consumption of transport over water approximately 17% of that of road transport and 50% of rail transport, it also has a high degree of safety and its noise and gas emissions are modest. This natural network is the only existing infrastructure with excess capacity and where congestion is limited (Inland Navigation Europe, 2014). In the US, total waterborne commerce has risen from about 1,500 million tons of goods in 1970 up to 2,600 million tons in 2006; due to the economic crisis it has lowered since then to a level of about 2,200 million tons in 2009 (U.S. Army Corps of Engineers, 2009). In a report prepared for the State of New York (Goodban Belt LLC, 2010), the potential of the New York Canal System for container-on-barge logistics is extensively described and an increased flow is expected after the Panama Canal expansion in 2015. The cargo volume transiting the Panama Canal is expected to grow on average 3% per year between 2005 and 2025, mainly due to an increase in container transport (Panama Canal Authority, 2006). In China, 88 million tons of freight passed the Three Gorges Dam in 2010; this is nearly 5 times the maximal annual volume reported before 2003 (ChinaDaily, 2011).

Many of these waterways (the Panama Canal, the Three Gorges Dam, inland waterways in Europe such as the Kiel canal (Luy, 2010), and many others) are accessible through sea locks and are often interrupted by river locks. Locks are needed to control the water level so that large and heavy ships can continue to access the corresponding waterways. At several waterway networks, congestion is expected to increase, yielding extra pressure on the locks. Examples are the New York State Canal System (Goodban Belt LLC, 2010) and the North Sea Canal region (van Haastert, 2003). Some locks are being expanded, such as locks on the Twente Canal in the Netherlands (Rijkswaterstaat, 2010). Also, new locks are being built, for example, the Deurganckdock lock at the harbor of Antwerp (Antwerp Port Authority, 2011), which will become the world's largest lock by volume.

These locks are bottlenecks for transportation over water, and hence, operating locks wisely contributes to the attractiveness of transportation over water and can help to avoid expensive infrastructural interventions. However, the algorithmic problem of how to operate a lock has not been studied broadly in the scientific literature. The purpose of this paper is to fill this gap. Our point

of view here is to see the lock as an entity providing a service to the ships. Then, it makes sense to identify a strategy for the lock that optimizes some criterion related to service (as we do in this paper). Another point of view would be that the lock announces times when ships can enter the lock in order to be transferred, and that the ships simply need to respect these times. Clearly, even in the latter model, there is still a decision to be made concerning these times.

1.1 Problem definition

We now give a formal description of a very basic situation that will act as our core problem: the lockmaster's problem. We first study this simplified problem; the results obtained for this problem will serve as a basis for dealing with more realistic settings later on. Consider a lock consisting of a single chamber. Ships that are travelling downstream arrive at the lock at given times. Other ships travelling upstream arrive at the lock, also at given times. Let $A = 1, \ldots, n$ represent the set of all ships. Let $t(a)$ be the given arrival time of ship $a \in A$, and $p(a)$ the arrival position of ship $a \in A$, with $p(a) = 0$ for a downstream arrival, and $p(a) = 1$ for an upstream arrival. For convenience, we assume that a total order $1 < 2 < \ldots < n$ is imposed on A so that ships are ordered by non-decreasing arrival time, i.e. $t(i) \leq t(i+1)$ for $i = 1, \ldots, n-1$. Note that multiple ships may have the same arrival time; the total order thus breaks these 'ties' arbitrarily. It is important to emphasize that we do not require ships to enter the lock in the order imposed on A, except in the cases (see sections 5.1 and 5.3) where we explicitly state this as a requirement.

Let T denote the *lockage duration*: this is the time needed for the water level to rise from the downstream level to the upstream level (or vice versa), plus the time needed to load and unload the lock (which is assumed to be constant in the basic problem). In other words, T measures the time that elapses between opening the lock such that ships can enter, and closing the lock after ships have left. Throughout the paper, we assume $T > 0$ as the problem becomes trivial for $T = 0$. We further assume that all data are integral. Our goal is to find a feasible lock-strategy that minimizes total waiting time of all ships. The waiting time of a ship is the length of the period that elapses between the ship's arrival time and the moment in time when the ship enters the lock. Thus, we need to determine at which moments in time the water level in the lock should start to go up (meaning at which moments in time downstream ships enter the lock and are lifted), and at which moments in time the water level in the lock should start to go down. For such a strategy to be feasible, (i) going-up moments and going-down moments should alternate, and (ii) consecutive moments should be at least T time-units apart.

This particular problem (to which we refer as the lockmaster's problem) is a simplified version of reality. However, we see this problem as a basic problem underlying any practical lock scheduling problem; and we show how to solve this basic problem by dynamic programming (DP) in Section 3. Practical problems obviously feature many properties that are absent in this basic problem. In Section 5 we give an overview of many such features and investigate the

complexity of these problem extensions.

The contributions of the present paper can be summarized as follows. We show that (1) there exists an $O(n^2)$ algorithm (see Section 3) for the lockmaster's problem; (2) this algorithm can be extended to solve variants with capacities, ship-dependent handling times, ship priorities, non-uniform lockage times, and settings with a limited number of lockages (Section 5).

In addition, we investigate the performance of several heuristics by running them on randomly generated instances that possess real-life characteristics (Section 6).

2 Literature

2.1 Lock scheduling

Lock scheduling has not been studied very thoroughly in the academic literature, although it has recently started to attract more attention. We mainly focus on the literature that considers scheduling a single lock. A relatively early work by Petersen & Taylor (1988) considers the Welland Canal in Canada. The authors present a dynamic programming algorithm for scheduling a single lock with unit capacity and extend this to obtain a heuristic result for the entire canal. A more recent paper (Nauss, 2008) deals with optimal sequencing in the presence of setup times and non-uniform processing times for the case where all arrival times are equal to zero. Smith et al. (2009) simulate the impact of decision rules and infrastructure improvements on traffic congestion along the Upper Mississippi River. Ting & Schonfeld (2001) use heuristic methods to study several control alternatives in order to improve lock service quality. Verstichel & Vanden Berghe (2009) develop (meta)heuristics for a lock scheduling problem where a lock may consist of multiple parallel, capacitated chambers of different dimensions and lockage times, making this problem at least as hard as a bin packing problem. The question of filling a lock with ships, i.e. the packing problem, is discussed by Verstichel et al. (2014a). Verstichel et al. (2014b) propose a mixed integer programming model to solve a generalized lock scheduling problem for instances with a limited number of ships. Recent work by Hermans (2014) considers the optimization problem of scheduling a single lock with a capacity that allows a single ship. A polynomial time procedure asserts feasibility with respect to given deadlines in $O(n^4 \log n)$ time; it can further be used to minimize the maximum lateness.

In a related problem, ships need to pass a narrow canal, and only a restricted number of wider areas is available where ships can pass each other. Ships need to wait in these areas and are arranged in convoys that transit in a one-way direction, see e.g. (Griffiths, 1995; Panama Canal Authority, 2006; Luy, 2010; Günther et al., 2011). For instances where all ships travel in the same direction, total waiting time is equal to zero, which is not necessarily the case in the lockmaster's problem.

2.2 Machine scheduling

Smith et al. (2011) relate traffic operations at a river lock with a variant of the job shop scheduling problem with sequence dependent setup times and a two-stage queuing process. The lockmaster's problem is more general since ships (i.e. jobs) have release dates and multiple ships can be locked together. The relation between the lock scheduling problem and classical machine scheduling is also observed in the literature. In fact, the lockmaster's problem introduced in this work is closely related to a batch scheduling problem. Batch scheduling involves a machine that can process multiple jobs simultaneously. Suppose that the basic lock scheduling problem only has downstream-bound ships (we will refer to this special case of the lockmaster's problem as the *uni-directional case*). The lock can be seen as a batching machine and the jobs are the arriving ships with release dates and equal processing times, and the flow time of a job is the waiting time of a ship. Following the notation of Baptiste (2000) this is problem $1|p-batch, b=n, r_i, p_i=p|\sum w_i F_i$. In words: the problem has a single parallel batching machine with unrestricted capacity ($b=n$), release dates on the jobs, and uniform processing times. The objective is to minimize the sum of weighted flow times ($\sum w_i F_i$), although we first discuss the problem where only unit weights are considered. Baptiste (2000) shows that this problem is polynomially solvable for a variety of objective functions. Cheng et al. (2005) developed an $O(n^3)$ algorithm for $1|p-batch, b=n, r_i, p_i=p|f$ where f can be any regular objective function. Condotta et al. (2010) show that feasibility of the same problem with bounded capacity and deadlines can be checked in $O(n^2)$, even for a setting with parallel batching machines. Ng et al. (2003) study a single machine serial batching scheduling problem with release dates and identical processing times. Machine setup only happens after arrival of the final job in a batch and there is a fixed setup time equal to s. The completion time of a batch is equal to the sum of the processing times of the jobs in the batch. This problem is equivalent to the uni-directional lockmaster's problem with $s=T$ and $p_i=p=0$, and can be solved in $O(n^5)$ by a dynamic programming algorithm described in Ng et al. (2003). Clearly, the lockmaster's problem is more general. Indeed, in case of both upstream and downstream-bound ships, we are dealing with two families of jobs, and only jobs of the same family can be together in a batch. Further, in our case, processing a batch of one family needs to be alternated by processing a (possibly empty) batch containing jobs of the other family; i.e. it is not possible to process two batches of the same family consecutively.

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

compatibility graph of the lockmaster's problem is the union of two cliques.

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

3 A polynomial time algorithm for the lockmaster's problem

In this section, we construct a graph such that the shortest path in the graph corresponds to a solution for the lockmaster's problem. We show that the corresponding shortest path algorithm runs in $O(n^3)$ time. In addition, we propose a more sophisticated implementation in Section 4, which results in an $O(n^2)$ algorithm.

3.1 Definitions and terminology

For clearness of presentation, let us describe some terminology. We define a *lock movement* as the act of bringing the water level down from its high point to its low point, or vice versa, plus loading and unloading the lock. Such a lock movement takes T time units, with $T>0$; thus, when a lock movement starts at time t, it finishes at $t+T$. A pair of lock movements is called *consecutive* when the starting time of the second lock movement equals the starting time of the first lock movement plus T. Clearly, after two consecutive lock movements starting at time t, the lock is back at the same position at time $t+2T$ as it was at time t. A set of ℓ lock movements is called *consecutive* when the pairs $(i, i+1)$ are consecutive, $i=1,\ldots,\ell-1$. Clearly, when a set of ℓ consecutive lock movements start at time t, it finishes at $t+\ell T$. Recall that each $a\in A$ has an associated *arrival time* $t(a)\in\mathbb{N}$ and an *arrival position* $p(a)\in\{0,1\}$ (1 for upstream and 0 for downstream).

Definition 1. *For each pair $a, b\in A$ with $p(a)=p(b)$, and $t(a)+2T<t(b)$, we define a block $B=(a,b)$ as the set of $\ell(B)$ consecutive movements that starts at $t(a)$ and ends at $t(a)+\ell(B)T$, where $\ell(B)$ is the largest even integer such that $t(a)+\ell(B)T<t(b)$.*

For each pair $a, b\in A$ with $p(a)\neq p(b)$, and $t(a)+T<t(b)$, we define a block $B=(a,b)$ as the set of $\ell(B)$ consecutive movements that starts at $t(a)$ and ends at $t(a)+\ell(B)T$, where $\ell(B)$ is the largest odd integer such that $t(a)+\ell(B)T<t(b)$.

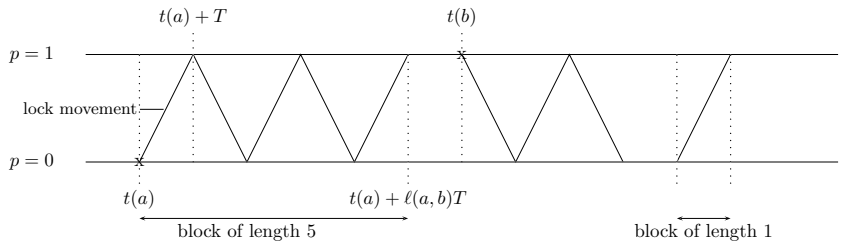

Figure 1: Lockmaster's problem: concepts

A geometric visualization of an instance and a corresponding solution is given in Figure 1, where upstream arrivals appear on the upper horizontal line and downstream arrivals on the lower horizontal line, and the schedule is represented by the lock movements between both lines.

We see a solution to the lockmaster's problem as a set of lock movements, each lock movement starting at some given time. Such a solution will be called a *schedule*. However, some schedules are more interesting than others. We first identify schedules that possess a certain structure.

Definition 2. *A schedule is called a* block-schedule *if it consists of a sequence of blocks $B_1, \ldots, B_s$ where $B_q = (a_q, b_q)$, $q = 1, \ldots, s$, and $b_q = a_{q+1}$ for each $q = 1, \ldots, s-1$; and with a final set of consecutive lock movements starting at $t(b_s)$, and ending not later than $t(n) + 3T$.*

Notice that the distinguishing feature of a block-schedule is that each movement starts at an arrival time or it directly follows the previous movement. Now, we are ready to state the following characteristic.

Lemma 1. *There is an optimal lock schedule that is a block-schedule.*

Proof. We first observe that the following properties characterize block-schedules:

Property 1. *Each lock movement either directly follows upon a previous lock movement, or starts upon some $t(a)$ while containing ship a.*

Property 2. *The length of a period in which there is no lock movement (a so-called* empty *period) is at most $2T$.*

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

It is easily verified that any schedule satisfying Properties 1-3 is a block-schedule, and any block-schedule satisfies these properties.

We now prove Lemma 1 by showing that any schedule not satisfying the above properties can be transformed into a schedule satisfying them without increasing the objective value. Consider some schedule for the lockmaster's problem not

satisfying Property 1. Let t denote the earliest time where a lock movement starts, such that t is neither the ending time of a previous lock movement, nor the arrival time $t(a)$ of some ship $a \in A$ contained in this lock movement. Since at $t - \epsilon$, with $\epsilon > 0$ and small, the lock is waiting to go up, and since no ships contained in the lock movement arrive between $t - \epsilon$ and t, we could have started this lock movement at $t - \epsilon$ without increasing the objective function value. Thus, we start the lock movement earlier in time, at either (1) the latest arrival time of a ship contained in the lock movement, or (2) the ending time of the preceding lock movement.

Next, consider a solution where no lock movement occurs during a period $S > 2T$. At the beginning of this period, two additional movements can be scheduled, reducing S by $2T$ time units. By repeating this step we end up only with empty periods shorter than $2T$ (Property 2).

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

We conclude that any schedule can be transformed in a block-schedule without deteriorating the objective value. This completes the proof. □

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

Property 4. *[FCFS] For each pair of ships $a, b \in A$ with $p(a) = p(b)$: if $a < b$, then ship a will leave the lock not later than ship b.*

Now, we describe an algorithm solving the lockmaster's problem in polynomial time. The basic idea is to build a directed acyclic graph $G = (V, E)$ with a given cost c_e for each $e \in E$. The arcs in the graph represent the blocks of a schedule, the situation before the first block or the final set of movements. After describing this graph, we argue that a path in this graph with a certain cost, corresponds to a block-schedule with a total waiting time equal to this cost, and vice versa. Thus, a shortest path corresponds to a solution to the lockmaster's problem.

3.2 Constructing the graph

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

There are five types of arcs, namely arcs leaving s, arcs entering t, so-called block1 arcs, block2 arcs, and 0-cost arcs; we now describe these arcs and the corresponding costs. See Figure 3 for a graphical representation, corresponding to the instance shown in Figure 2, assuming $T = 30$.

There is an arc from s to a single node from each layer $L(a)$, $a \in A$, say node a_k, with $k \in \{0, \ldots, n\}$. The value of k is determined by the number of

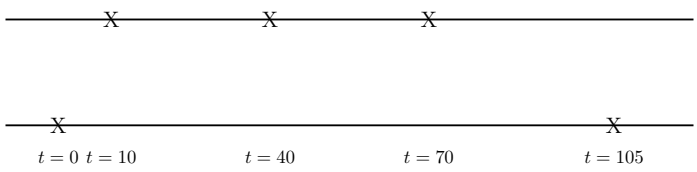

Figure 2: Lockmaster's problem: example instance

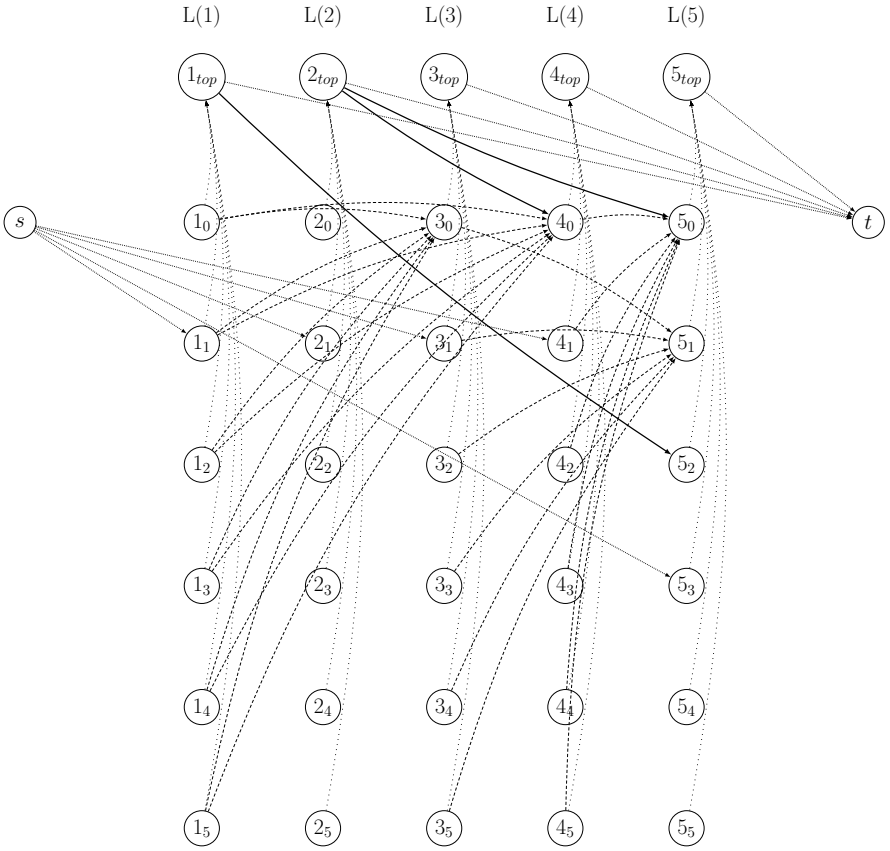

Figure 3: Lockmaster's problem: the graph

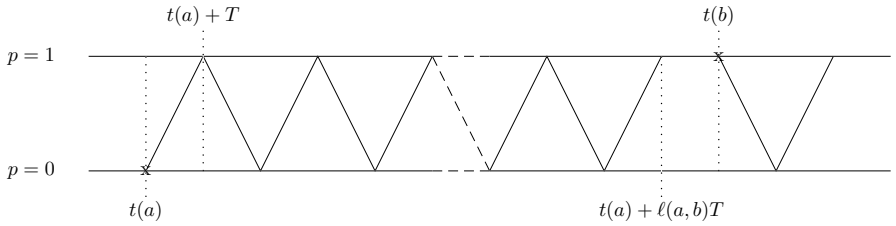

Figure 4: Waiting times covered by block (a, b) of length at least 2

ships arriving in $(-\infty, t(a)+T]$ at position $1-p(a)$. This arc represents the situation where the first block of a schedule starts at time $t(a)$ and position $p(a)$. Accordingly, the cost of this arc equals the waiting time accumulated at time $t(a)$ of all ships arriving in $(-\infty, t(a)]$ at position $p(a)$, plus the waiting time accumulated at time $t(a)+T$ of all ships arriving in $(-\infty, t(a)+T]$ at position $1-p(a)$.

There is an arc from a top node a_{top} from each layer $a \in A$ to node t. The cost of this arc equals the waiting time of all ships arriving in $[t(a), t(n)]$ at position $p(a)$, and in $[t(a)+T, t(n)]$ at position $1-p(a)$, in a solution where at time $t(a)$ a set of consecutive movements starts, serving all these ships.

We now describe the block2 arcs. For each block $B=(a,b) \in A \times A$ with $\ell(B) \geq 2$, there is an arc from node a_{top} to node $b_l \in L(b)$, where l equals the number of ships that arrive at position $1-p(b)$ in the interval $(t(a)+(\ell(B)-1)T, t(b)+T]$. The cost of such an arc equals the waiting time accumulated in block B at time $t(b)+T$, of, in case $p(a)=p(b)$, all ships arriving in $(t(a), t(b)]$ at position $p(a)$, and in $(t(a)+T, t(b)+T]$ at position $1-p(a)$; in case $p(a) \neq p(b)$, all ships arriving in $(t(a), t(b)+T]$ at position $p(a)$, and in $(t(a)+T, t(b)]$ at position $1-p(a)$. These are the block2 arcs. Figure 4 illustrates a block (a, b) of length at least 2. It can be seen easily that all ships arriving in $p=1$ in $(t(a)+T, t(a)+3T]$ are moved at time $t(a)+3T$. Similarly, all ships arriving in $p=1$ in $(t(a)+(\ell(a,b)-2)T, t(b)]$ are moved at time $t(b)$. Finally, notice that ships arriving in $p=0$ in $(t(a)+(\ell(a,b)-1)T, t(b)+T]$ are moved at time $t(b)+T$.

We now describe the block1 arcs. For each block $B=(a,b) \in A \times A$ with $\ell(B)=1$, there is an arc from each node a_k $(k=0,\ldots,n)$ to some node b_l from layer $L(b)$ where l equals the number of ships that arrive at position $1-p(b)$ in the interval $(t(a), t(b)+T]$. The cost of such an arc consists of two parts, first, the waiting time accumulated at time $t(b)+T$ of all ships arriving in $(t(a), t(b)+T]$ at position $p(a)$, and in $(t(a)+T, t(b)]$ at position $p(b)$; and second, $k \times (t(b)-t(a)-T)$. Figure 5 illustrates a block (a, b) of length 1. Ships arriving in the time interval represented by the dashed section at $p=0$, are transported by a movement starting at time $t(b)$. Since $t(b) > t(a)+T$, we need to take this additional waiting time into account, which is achieved by the second term

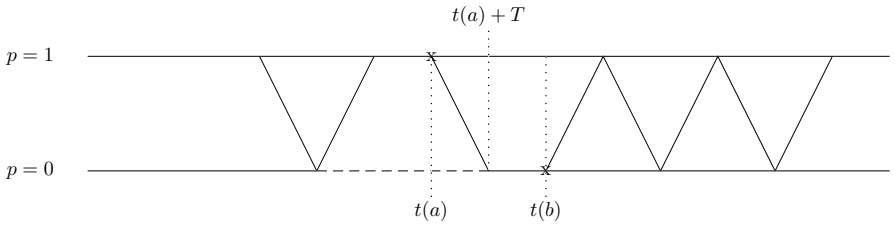

Figure 5: Waiting times covered by block (a, b) of length 1

described above. Note that, by construction of the block2 arcs, the index k of a_k corresponds precisely to the number of these waiting ships for block2 arcs ending at a_k.

Finally, within each layer $L(a), a \in A$, there is an arc with cost 0 that goes from each node $a_k, k = 0, \ldots, n$, to node a_{top}; these are the 0-cost arcs. This completes the description of graph G. We now prove a lemma that establishes the correspondence between block-schedules and paths in G.

Lemma 2. *An s, t path in G corresponds to a block schedule and vice versa.*

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

Theorem 1. *The lockmaster's problem can be solved in $O(n^3)$ by a straightforward implementation of a shortest path algorithm on the acyclic graph G.*

Proof. The previous Lemma, combined with Lemma 1 and the observation that the graph G is acyclic, and contains $O(n^3)$ arcs, imply this result. □

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

4 A faster algorithm for the lockmaster's problem

We describe here a more efficient implementation that uses the structure in the graph G to solve the lockmaster's problem. The resulting complexity is $O(n^2)$. The method used to obtain this speed-up shows similarities to the approaches proposed by Wagelmans et al. (1992), Federgruen & Tzur (1991), and Aggarwal & Park (1993), where an improvement procedure is described to solve the well-known lot sizing problem. We use a specific forward labelling algorithm, where we label the nodes layer by layer, and within a layer, we start with a_0, then a_1, up to a_n, and finally a_{top}.

Define $sp(a_k)$ as the shortest path length from s to node a_k, with $k = 0, \ldots, n$. Further, define $sp^{a_k}(b_i)$ as the shortest path length to node b_i, with $i = 0, \ldots, n$; $b = a+1, \ldots, n$, when the node visited just before b_i is a_k. Note that the arc (a_k, b_i) is a block1 arc in G, which is defined only when $t(a) + T < t(b)$ and $p(a) \neq p(b)$.

There are $O(n^2)$ arcs leaving nodes from layer $L(a)$, $O(n)$ block2 arcs and $O(n^2)$ block1 arcs. A shortest path to some node b_i has as last arc either a block1 arc, or some arc that is not a block1 arc. It follows that, for each $b_i \in L(b)$,

$$sp(b_i) = \min \left\{ \begin{array}{c} \min\limits_{\substack{a<b \\ k=0,\ldots,n}} \left\{ sp^{a_k}(b_i) \right\}, \min\limits_{a<b} \left\{ sp^{a_{top}}(b_i) \right\}, sp^{s}(b_i) \end{array} \right\}.$$

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

For a given b_i, it holds that $sp^{a_k}(b_i) = sp(a_k) + c(a_k, b_i)$, whereby $c(a_k, b_i) = w(a, b) + k(t(b) - t(a) - T)$ is the length of the block1 arc (a_k, b_i) in G. The term $w(a, b)$ represents the waiting time accumulated at time $t(b) + T$ of all ships arriving in $(t(a), t(b) + T]$ at position $1 - p(b)$ and the waiting time of all ships arriving in $(t(a) + T, t(b)]$ at position $p(b)$; this is a constant over all a_k, b_i for which a block1 arc (a_k, b_i) exists. Note that for each a_k, there is at most one b_i from layer $L(b)$ for which an arc (a_k, b_i) exists. We argue that finding all $sp^{a_k}(b_i)$ for $k = 0, \ldots, n$ and for $b = a+1, \ldots, n$ (i is fixed, given a_k and b), corresponds to finding the minimum value of at most n linear functions ($\forall k = 1, \ldots, n$) for no more than n inputs ($\forall b = a+1, \ldots, n$).

Lemma 3. *Given m linear functions $f_i(x) = \alpha_i \cdot x + \beta_i$ with $\alpha_i, \beta_i \in \mathbb{R}$ for each $i = 1, \ldots, m$, $\alpha_i \leq \alpha_{i+1}$ for each $i = 1, \ldots, m-1$, and an ordered finite set Q. Then the indices $\arg\min\{f_i(q) \mid i = 1, \ldots, m\}$ for each $q \in Q$ can be found in $O(m + |Q|)$.*

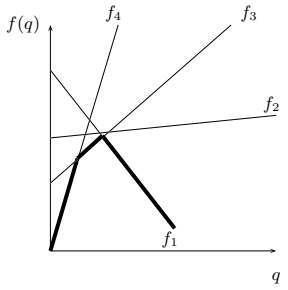

Figure 6: Lower envelope

Proof. It is clear that we may assume w.l.o.g. that no two of the given functions are identical. We start by eliminating dominated functions. We say that f_i is dominated by f_j, $j < i$, if $\beta_j \leq \beta_i$ because then $f_j(q) \leq f_i(q)$ for each $q \geq 0$. Henceforth, we assume that $\beta_i > \beta_{i+1}$ for each $i = 1, \ldots, m-1$ in the following.

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

Finally, by scanning through the sorted set of intersections of linear functions and through values in Q, we find $\arg\min\{f_i(x) \mid i = 1, \ldots, m\}$ for each $q \in Q$. For a graphical representation of the lower envelope, see Figure 6.

It remains to show that this procedure runs in linear time. Eliminating dominated functions takes $O(m)$. During the construction of $\mathcal{L}$, each function is added at most once and each function is deleted at most once. The decision whether to add the next function at the end of $\mathcal{L}$ or remove the currently last function from $\mathcal{L}$ can be taken in constant time. Thus, constructing $\mathcal{L}$ takes $O(m)$ and, consequently, finding $\arg\min\{f_i(q) \mid i = 1, \ldots, m\}$ for each $q \in Q$ takes $O(m + |Q|)$. □

We now use Lemma 3 and consider all block1 arcs leaving $L(a)$, for a given $a \in A$. Consider the linear functions $sp^{a_k}(b_l) = sp(a_k) + w(a,b) + k(t(b) - t(a) - T)$, for $k = 0, \ldots, n$ and for $b = a+1, \ldots, n$. As mentioned before, $w(a,b)$ is a constant for any given a and b. We thus exclude it from the linear

```
construct graph G as described above
sp(a_k) ← ∞ for all a ∈ A and 0 ≤ k ≤ n
for a ∈ A do
  consider block (s, a) and update sp(a_k) for k corresponding to (s, a)
end for
for a = 1, ..., n do
  set sp(a_top) = min {sp(a_k) | 0 ≤ k ≤ n}
  determine the lower envelope for all block1 arcs (a_k, b_i)
       with k = 0, ..., n and b = a + 1, ..., n (i follows from a_k and b)
  for b = a + 1, ..., n do
    if l(a, b) == 1 then
      update sp^{a_k}(b_i) using the lower envelope
    else
      update sp^{a_k}(b_i) using sp(a_top)
    end if
  end for
end for
```

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

functions to be considered for the lemma, and add this constant later. Let $Q = \{t(b) - t(a) - T \mid b = a + 1, \ldots, n\}$ and $f_k(q) = sp(a_k) + q \cdot k$ for each $0 \leq k \leq n$. Thus, referring to Lemma 3, $\alpha_k = k$, $\beta_k = sp(a_k)$, and $q \in Q$. In Lemma 3 it is assumed that the α_i and the set Q are ordered, which is obviously the case. Now, $\arg\min\{sp^{a_k}(b_i) \mid 0 \leq k \leq n\}$ for each b_i determines the shortest path to b_i having a block (a_k, b_i) of length one as last part. Note that its length is $\min\{f_k(q) \mid 0 \leq k \leq n\} + w(a, b)$. Hence, we can update all $sp(b_i)$ taking into account all block1 arcs leaving layer $L(a)$ in $O(n)$ time.

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

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

5 Extensions

It can be argued that the basic problem defined in Section 1, due to different assumptions regarding the input, ignores a number of issues regarding practical lock operation. We now proceed by showing how the procedure described in Section 3 can be extended towards a closer approximation of reality.

5.1 Capacity

Section 3 did not take any capacity restrictions into account. We now discuss two different settings with capacity restrictions. In one setting, the lock can accommodate at most a given number of ships; in another setting each ship has a size, the total size of ships within a lock should then not exceed the given lock's

size. A fundamental difference with the basic lockmaster's problem is that ships may be "left behind". Thus, ships that lay waiting may not all be transferred in the same lock movement.

When the number of ships that may be contained within the lock at any given time is bounded by a constant, a similar procedure to Section 3 may be used.

Theorem 3. *The lockmaster's problem with a bound on the number of ships in the lock is solvable in* $O(n^4)$.

Proof. Suppose that the lock can accommodate at most c ships at once. We can reformulate Property 3 as follows: the final lock movement does not end later than $(1 + \lceil \frac{n}{c} \rceil)2T$. For this generalization, Lemma 1 remains true. Indeed, Properties 1 and 2 remain trivially true; moreover, the ships being identical (except for their arrival time) implies that there exists an optimal schedule that satisfies Property 4.

Definition 3.

- *For each* $a \in A$, *let* $U_a = \{b \in A \mid p(b) = 1, b < a, t(b) \leq t(a) - |p(b) - p(a)|T\}$.
- *For each* $a \in A$, *let* $D_a = \{b \in A \mid p(b) = 0, b < a, t(b) \leq t(a) - |p(b) - p(a)|T\}$.

Thus, the set U_a (D_a) contains all ships b for which $b < a$, that arrive upstream (downstream), not later than $t(a)$ if ship a arrives upstream (downstream), and not later than $t(a) - T$ if ship a arrives downstream (upstream).

We now design the following directed graph $G = (V, E)$ in order to represent the problem. In contrast to the graph used for the lockmaster's problem, here, each arc directly corresponds to a set of ships transferred, and there is no need to distinguish block1 and block2 arcs. We have $V = \{s, t\} \cup V^A$ with $V^A = \{(a, u, d) \mid a \in A, u \in U_a, d \in D_a\}$. A node (a, u, d) corresponds to a block starting at $t(a)$ and position $p(a)$ while u and d represent the latest (with respect to the ordering in A) served downstream- and upstream-bound ship, respectively. It is easily seen that whenever a ship a does not enter a lockage starting not earlier than $t(a)$ from position $p(a)$ while sufficient lock capacity is available to serve this ship, the solution is suboptimal. In combination with Property 4 it follows that, within each block, each ship is served by the next appropriate lockage with sufficient capacity, and it is thus known when each ship is handled. We thus know which ships u' and d' become the latest ships served upstream and downstream, respectively. We record this observation by writing $u' = f_{\text{up}}((a, b), u)$ and $d' = f_{\text{down}}((a, b), d)$ for each block (a, b), $u \in U_a$, $d \in D_a$.

Definition 4. *We say that two nodes* (a, u, d) *and* (b, u', d') *are* compatible *if* (a, b) *is a block and if* $u' = f_{up}((a, b), u)$ *and* $d' = f_{down}((a, b), d)$.

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

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

- $E^B = \{(v, v') \mid v \text{ and } v' \text{ are compatible}\}$, and
- $E^t = \big\{(v, t) \mid v = (a, u, d) \in V^A\big\}$.

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

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

Then, a path from s to t represents a feasible lock schedule and the shortest path represents the lock schedule having minimum total waiting time of ships. G is acyclic and contains $O(n^3)$ nodes, each node being source of at most $O(n)$ arcs. Furthermore, we can determine u', d', and total waiting of ships according to all arcs emerging from $v \in \{s\} \cup V^A$ in $O(n)$ time. The total time complexity for this algorithm is thus $O(n^4)$. □

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

Theorem 4. *The lockmaster's problem with a (scalar) bound on the size of the lock is NP-hard.*

Proof. We provide a proof by reduction from 3-PARTITION. In an instance of 3-PARTITION, we are given an integer B and a set A consisting of integers a_i $(i = 1, \ldots, 3n)$ subject to $\frac{B}{4} < a_i < \frac{B}{2}$ for each i. The question is whether A can be partitioned into n triples such that the sum of integers in each triple equals B. This problem is known to be NP-complete. We now construct an instance of the lockmaster's problem where the size of a ship is encoded by the number of the 3-PARTITION instance, all ships arrive at $t = 0$ in position $p = 1$, and the lock's size encodes the value B to be met by each triple. Clearly, if the instance of 3-PARTITION has a solution, then there exists a lock schedule where n lockages each serve exactly 3 ships. Each of these lockages is followed by an empty lock movement that returns the lock to position $p = 1$, resulting in a solution with a total waiting time of $3n(n-1)T$. On the other hand, any solution with a total waiting time of $3n(n-1)T$ such that at most 3 ships can be handled in each subsequent $2T$ interval, will transfer all ships in the first n lockages starting from position $p = 1$. Consequently, there must exist n subsets of ships so that each subset fills the lock completely, and thus a solution exists to the instance of 3-PARTITION. □

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

Theorem 5. *Under a FCFS policy, the lockmaster's problem with a bound on the size of the lock is solvable in $O(n^4)$.*

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

Notice that although in the instance constructed in the proof of Theorem 4 all ships arrive at the same time, it is not true that every sequence of ships respects FCFS; only the sequence that is compatible with the given order on A adheres to the FCFS policy. This explains why Theorems 4 and 5 are not contradictory.

5.2 Ship priorities

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

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

Theorem 6. *The lockmaster's problem with objective to minimize total weighted waiting time is solvable in $O(n^3)$.*

Notice that when the ships are weighted and the capacity of the lock is limited, the algorithm described in Section 5.1 might fail to find an optimal solution. Earlier, specifying a block would implicitly specify the set of ships that are transferred in this block. This, however, need then no longer be the case;

although the optimal solution can still be seen as consisting of blocks, it is not clear which particular ships are transferred during which movements in a block. When considering the uni-directional case, Baptiste's algorithm (Baptiste, 2000) (see Section 2) yields a polynomial time procedure.

5.3 Handling times

In practice, placing a ship into a lock takes a certain amount of time, such that the total time a ship spends in the lock may depend on the other ships present. The required handling time per ship may be constant, which is approximately the case for inland ships. For sea ships, on the other hand, the handling time depends on the size and manoeuvrability of the ship. In Smith et al. (2011), a lock scheduling problem is described where handling times are not only ship-dependent, but also sequence dependent. In this section, we consider the extension where each ship $a \in A$ requires a certain, integral, handling time h_a. We model this by modifying the lockage time so that it is no longer constant, but depends on the ships that are present in the lock movement: $T_j = T + \sum_{a \in A_j} h_a$, where A_j refers to the set of ships transferred in the j-th lock movement, and T_j denotes the corresponding lockage duration, $j = 1, 2, \ldots$.

Observe that, due to possibly distinct h_a values, it may be optimal to let a ship with a relatively large handling time wait while handling subsequent arrivals first. Thus optimal solutions may involve the overtaking of ships, and hence violate Property 4. We show that the lockmaster's problem with ship dependent handling times is NP-hard. However, as mentioned earlier, the FCFS policy is a reasonable additional requirement when dealing with lock scheduling problems. Also notice that, even when solutions must satisfy FCFS, it might be beneficial to let a ship with a relatively large handling time wait, so that the lock is able to move quickly to the other side and transfer ships that have arrived there. Nevertheless, we show that for this case a polynomial time dynamic programming approach exists. Note that this also applies to the setting where all handling times are equal, i.e. the case where $h = h_a$ for all $a \in A$.

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

We first show that the general setting for the lockmaster's problem with

handling times is NP-hard.

Theorem 7. *The lockmaster's problem with ship-dependent handling times is NP-hard.*

Proof. We provide a reduction from 3-PARTITION (see the proof of Theorem 4 for the definition). For any given instance of 3-PARTITION, we construct a corresponding instance of the lockmaster's problem with ship-dependent handling times. We argue that solving the latter instance to optimality allows to decide the question of the 3-PARTITION instance.

The lockmaster's instance is constructed as follows: Let $T = 1$, although it can be seen that an instance can be constructed for arbitrarily chosen T. At time $t = 0$, a set of $3n$ ships arrives on the downstream side, with for each ship a handling time $h_i = a_i$ $(i = 1, \ldots, 3n)$. At each of the times $t = B + T, t = 2B + 3T, \ldots, t = n(B + 2T) - T$, a set of $3(B + 2T)n^2$ ships arrives on the upstream side. Each of these ships has zero handling time, i.e. $h_j = 0$ for $j = 3n + 1, \ldots, 3n + 3(B + 2T)n^3$. The question is the following: "does there exist a solution with total waiting time not greater than $Q \equiv 3(B + 2T)n(n - 1)/2$?" This completes the description of the instance of the lockmaster's problem with handling times. Figure 7 provides a graphical illustration of this instance.

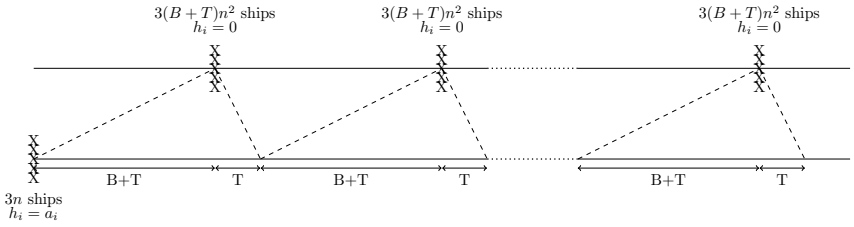

Figure 7: Illustration of the instance for the proof of Theorem 7

We now argue that the existence of a solution to the lockmaster's instance with total waiting time not greater than Q implies a 'yes' answer to the 3-PARTITION question. Observe that since all data are integral, we can assume that the lock moves only at integral moments in time. Then from $Q < 3(B + 2T)n^2$ it follows that any solution with value no greater than Q must handle all upstream ships at their arrival time. Hence, for all such solutions, the lock moves from the upstream to the downstream side at times $t = B + T, t = 2B + 3T, \ldots, t = n(B + 2T) - T$. The lock thus has n intervals of size $B + T$ at the downstream side to handle the ships that arrive at $t = 0$. Because the duration T cannot be avoided regardless of the number of ships in the lock, it follows that the sum of handling times of all ships in each upward lockage can be at most equal to B. Since $\frac{B}{4} < a_i$, it is impossible to handle four ships in such an interval. As a result, the only way to achieve a total waiting time not greater than $\sum_{i=1}^{n} 3(n - i)(B + 2T)$, is

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

Note that when the waiting time for a ship is defined according to the alternative definition described in Section 5.3, we can easily modify the proof so that the theorem remains valid. Indeed, we may choose $T > \sum_{i=1}^{3n} h_i$, increase the number of ships arriving on the upstream side at $t = B+T, t = 2B+3T, \ldots,$ $t = n(B+2T)-T$ by $2\sum_{i=1}^{n} h_i$, and let $Q \equiv 3(B+2T)n(n-1)/2+2\left(\sum_{i=1}^{3n} h_i\right)$. Observe that because 3-PARTITION is NP-complete in the strong sense, we may assume that the values a_i are bounded by a polynomial in n; the size of the corresponding lockmaster's instance thus also remains polynomial in n. Because the value for T was chosen to be sufficiently large, any solution where an upwards lockage contains two or less ships necessarily has an objective value larger than Q. Each lockage starting in position $p = 0$ in a solution with value not greater than Q must thus contain exactly 3 ships. It then follows that, in all such solutions, a waiting time of $2\left(\sum_{i=1}^{3n} h_i\right)$ is incurred due to ships being simultaneously present within the lock. The remaining argumentation for the proof remains unchanged since the redefinition of Q reflects this increase in waiting time.

Similar to the setting with capacity and arbitrary ship sizes (Section 5.1), which is also NP-hard in general, we observe that a polynomial time procedure exists when restricting the solutions to those which adhere to the FCFS policy.

Theorem 8. *The lockmaster's problem with arbitrary handling times under a FCFS policy can be solved in $O(n^{10})$ time.*

Proof. Observe that the arguments used to establish Lemma 1 in Section 3 also apply to the case of arbitrary handling times. Hence properties 1 and 2 remain valid in this setting.

We now construct a graph $G = (V \cup \{s,t\}, E)$ as follows. For each ship $a \in A$, and for each possible number of ships waiting upstream at time $t(a)$ (referred to as w_u), and for each possible number of ships waiting downstream at time $t(a)$ (referred to as w_d), we create a node (also called state) $v = (a, w_u, w_d)$ in V. Notice that a is not included in either w_u or w_d. To define the edges of G, we consider each pair of states (v, v') with $t(a') > t(a) + T$ if $p(a) \neq p(a')$ or with $t(a') > t(a) + 2T$ if $p(a) = p(a')$. It is important to realize that, when given two states v and v', it is exactly known *which* ships are transferred when moving from state v to state v'. Indeed, the FCFS assumption, together with

the number of ships waiting at $t(a)$ and $t(a')$ directly reveals which ships have been transferred. We will denote this set of ships by $A(v, v') \subseteq A$.

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

Lemma 4. *Given a pair of states $v = (a, w_u, w_d)$ and $v' = (a', w'_u, w'_d)$, any feasible handling strategy consists of the same number of lock movements.*

Proof. Since we know v and v', we know which ships are transferred by any handling strategy, and hence we know the total handling time: $H(v, v') = \sum_{a \in A(v,v')} h_a$. By Property 2 above, we also know that the ending time of the last movement of the strategy cannot exceed $t(a')$, nor can it end earlier than $t(a') - 2T$. Thus, in case $p(a) = p(a')$, we look for an even number of movements, called $m(v, v')$, that satisfies:

$$t(a') - 2T < t(a) + H(v, v') + m(v, v')T \leq t(a').$$

Similarly, in case $p(a) \neq p(a')$, we look for an odd number of lock movements $m(v, v')$ satisfying the equation above. Due to Property 2, the value of $m(v, v')$ is unique and the lemma follows. □

For all pairs v, v' for which a number $m(v, v')$ can be computed, we have an edge in G. Further, there is an edge from node s to each state v, and there is an edge from each state v to node t.

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

The cost of a handling strategy consists of the waiting time of relevant ships. A ship b is relevant when, at time $t(a)$, it has not yet been transferred and at time $t(a')$ it has arrived, (i.e. when $t(b) \leq t(a')$). For each relevant ship b, its waiting time equals the time that elapses between $max(t(a),\ t(b))$; until the minimum of the starting time of the lock movement containing b, and $t(a')$.

Let us now show how to compute the minimum total waiting time of the optimal handling strategy from v to v'. This will become the cost of the arc from state v to v' in the graph G. Recall that $A(v, v')$ is the set of ships transferred; we write $A(v, v') = A^u(v, v') \cup A^d(v, v')$, where $A^u(v, v')$ ($A^d(v, v')$) refers to those ships in $A(v, v')$ that arrive on the upstream (downstream) side. Given an

integer k, let $t_u(k)$ and $t_d(k)$ refer to the arrival time of the k'th ship in $A^u(v, v')$ and $A^d(v, v')$ respectively.

Given v and v', we construct a graph H. There is a node in H for each state (k_u, k_d, m); k_u ranges from $0, \ldots, |A^u(v, v')|$, k_d ranges from $0, \ldots, |A^d(v, v')|$, and $m = 1, \ldots, m(v, v')$. Note that $m(v, v')$ can be uniquely determined as a result of Lemma 4. Thus, a state (k_u, k_d, m) represents the situation that the first k_u upstream ships from $A(v, v')$, as well as the first k_d downstream ships from $A(v, v')$ have been transferred using m lock movements. In addition, we add a starting node $(0, 0, 0)$. Let us now consider the edges in H. For any vertex $w = (k_u, k_d, m)$, we find all possible states $w' = (k'_u, k'_d, m + 1)$ that can be reached by the next lock movement. Notice that this includes an empty lock movement, i.e. state $(k_u, k_d, m + 1)$. Also, a single lock movement cannot serve both upstream and downstream ships: thus, only states with either $k'_u = k_u$ or $k'_d = k_d$ can be reached. More formally, we associate to each state a so-called ending time $T(k_u, k_d, m)$, which can be computed as follows:

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

where $A_{k_u, k_d}(v, v')$ corresponds to the first k_u upstream ships, and the first k_d downstream ships from $A(v, v')$. Observe that $T(0, 0, 0) = t(a)$.

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

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

Lemma 5. *All handling strategies where two consecutive empty lock movements are followed by a nonempty movement can be excluded without impacting the optimality of the resulting solution.*

Proof. Assume w.l.o.g. that the non-empty movement starts on the downstream side. Notice that the handling strategy is non-optimal if any ship moved by the nonempty movement arrives before the first empty movement. We thus assume

the existence of at least one downstream arrival during the empty movements. It is then easy to see that there is an optimal block schedule with a block starting at that arrival, and thus excluding the two empty movements. □

Note that for all m larger than $2|A(v,v')|+1$, either two consecutive empty movements are followed by a nonempty movement, or all ships in $A(v,v')$ have been moved. The values for m that we need to consider is thus bounded by $O(n)$ so that we find a shortest path in H considering only $O(n^3)$ nodes and $O(n^4)$ arcs.

Correctness of the theorem thus follows from the fact that we argued that an optimal solution exists that displays the structure of Lemma 1. Next, we showed how to compute, for each possible pair of states, a handling strategy with minimum cost. Since any s-t path in G represents a feasible lock schedule, the shortest path in this graph corresponds to the optimal lock schedule. With respect to the complexity of the procedure we observe by Lemma 5 that the number of states to be considered in G equals $O(n^3)$, and hence the number of edges equals $O(n^6)$. Since a shortest path in H can be found in $O(n^4)$ steps, we have a polynomial time bound of $O(n^{10})$. □

5.4 Non-uniform lockage duration

It is not uncommon that lockage times for going up (T_u) and down (T_d) differ. Indeed, upstream going ships might need more time to enter the lock compared to downstream going ships, due to the current. We argue that the procedure for the lockmaster's problem outlined in Section 3 is easily adjusted to take this into account.

Theorem 9. *The lockmaster's problem with a lockage duration that depends on the position of the lock is solvable in $O(n^3)$.*

Proof. We redefine a block $B(a,b)$ from Section 3 as follows.

Definition 5.

- *For each $a,b \in A$ with $p(a)=0\neq p(b)$ and with $t(a)+T_u<t(b)$, a block $B(a,b)$ is the set of $l(B)$ consecutive movements that starts at $t(a)$ and ends at $t(a)+\lceil\frac{l(B)}{2}\rceil T_u+\lfloor\frac{l(B)}{2}\rfloor T_d$; with $l(B)$ the largest odd integer such that $t(a)+\lceil\frac{l(B)}{2}\rceil T_u+\lfloor\frac{l(B)}{2}\rfloor T_d\leq t(b)$.*
- *For each $a,b \in A$ with $p(a)=1\neq p(b)$ and with $t(a)+T_d<t(b)$, a block $B(a,b)$ is the set of $l(B)$ consecutive movements that starts at $t(a)$ and ends at $t(a)+\lceil\frac{l(B)}{2}\rceil T_d+\lfloor\frac{l(B)}{2}\rfloor T_u$; with $l(B)$ the largest odd integer such that $t(a)+\lceil\frac{l(B)}{2}\rceil T_d+\lfloor\frac{l(B)}{2}\rfloor T_u\leq t(b)$.*
- *For each $a,b \in A$ with $p(a)=p(b)$ and with $t(a)+T_u+T_d<t(b)$, a block $B(a,b)$ is the set of $l(B)$ consecutive movements that starts at $t(a)$ and ends at $t(a)+\frac{l(B)}{2}(T_u+T_d)$; with $l(B)$ the largest even integer such that $t(a)+\frac{l(B)}{2}(T_u+T_d)\leq t(b)$.*

It is not difficult to verify that all results from Section 3, mutatis mutandis, apply to this setting. □

5.5 Water usage

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

Theorem 10. *The lockmaster's problem with a bound Q on the number of lock movements is solvable in $O(Q^2n^4)$.*

Proof. Let Q refer to the maximum number of allowed lockages. Notice that in this situation Lemma 1 no longer holds. Indeed, a lock movement will no longer occur at least every $2T$ interval. However, it does hold that lock movements are either consecutive or start upon arrival time of a ship; and the final lock movement ends no later than $t(n) + 3T$. We now define a *bounded block* $B = (a, b, l(B))$ for each $a, b \in A, l(B) \leq Q$ with $t(a) + l(B)T < t(b)$, as a set of $l(B)$ consecutive movements starting in $t(a)$, with $l(B)$ even if $p(a) = p(b)$, and $l(B)$ odd if $p(a) \neq p(b)$. It must hold that $t(a) + l(B)T < t(b)$.

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

If we define U_a and D_a as in Section 5.1, we may design the following directed graph $G = (V, E)$ in order to represent the problem. We have $V = \{s, t\} \cup V^A$ with $V^A = \{(a, u, d, q) \mid a \in A, u \in U_a, d \in D_a, 0 \leq q \leq Q\}$. A node (a, u, d, q) corresponds to a bounded block starting at $t(a)$, while u and d represent the latest ships served coming from upstream and downstream, respectively, and q denotes the number of finished lock movements. It holds that:

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

Definition 6. *We say that two nodes (a, u, d, q) and (b, u', d', q') are* compatible *if (a, b) is a block and if $u' = f_{up}((a, b, l(a, b)), u)$ and $d' = f_{down}((a, b, l(a, b)), d)$, and $q' = q + l(a, b)$.*

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

- $E^s = \big\{(s, v) \mid v = (a, \emptyset, \emptyset, 0) \in V^A\big\}$,
- $E^B = \{(v, v') \mid v \text{ and } v' \text{ are compatible}\}$, and

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

The meanings of nodes and arcs are analogous to the ones in Section 5.1. The only difference is that the number of lock movements is encoded in nodes here and arcs represent bounded blocks instead of blocks. Graph G now contains $O(n^3Q)$ nodes. Note that each node is source of $O(Qn)$ arcs. We can determine $u'((a,b,l(a,b)),u)$ and $d'((a,b,l(a,b)),d)$, and total waiting time of ships according to all arcs emerging from $v \in \{s\} \cup V^A$ in $O(Qn)$ time.

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

6 Computational study

Is it worth the effort to solve instances of the lockmaster's problem to optimality? Or, are heuristics sufficient to achieve near-optimal solutions? In this section we answer this question by performing computational experiments. We consider the basic problem setting as well as a number of extensions and compare the optimal solution to a number of straightforward heuristics.

6.1 Instances and problem setting

As far as we are aware, the only publicly available instances are maintained by Verstichel & Vanden Berghe (2009). We report results for these instances below. As these instances do not contain all information needed for the extensions we cover here, we also generate new instances for the experiments below, simulating a realistic arrival process. We sample the waiting time between subsequent arrivals from a memoryless distribution, i.e. the arrival probability for a ship is unrelated to the elapsed time since the last arrival. To obtain integral arrival times, we use a geometric distribution which allows for simultaneous arrivals. The geometric distribution can be seen as the discrete analogue of the negative exponential distribution. The time unit is one minute. All instances consider a time horizon of 24 hours. The lockage duration is assumed to be 30 minutes, a typical value similar to that reported by Smith et al. (2009) for 'single' lock operations. The only remaining parameter to decide upon is the parameter p that specifies the geometric distribution. This parameter corresponds to the arrival probability as follows: if we let X denote a random variable representing the number of ships that arrive at any given time in the instance, the following holds: $P(X \geq n) = p^n$ for all $n \in \mathbb{N}$. Each generated instance can thus be considered as a random realization of a 'typical day' where the arrival probability is specified by the value of p. We report here the results averaged over 25 random instances with equal p. Rijkswaterstaat has kindly provided a dataset with arrival information for the three parallel locks of Terneuzen, the Netherlands. Regression on the interarrival times shows that the geometric distribution assumed above

corresponds well to reality. Furthermore, the arrival probability of $p = 0.1$ used in the simulations below reflects the arrival process observed at the locks of Terneuzen. In addition to the arrival times, each arriving ship is also assigned a priority value and a handling time for the extensions below. We consider the following problem settings:

1. **Basic**: The basic settings described in Section 3. This considerably simplified problem serves as the starting point for all cases below. Further, it acts as a reference to compare the relative performance of the different solution methods.

2. **Capacity**: The extension covered in Section 5.1. We assume that the number of ships in the lock is limited for any lock movement. We arbitrarily set the bound equal to 3 for all capacitated instances below. Separate simulations with capacity values of 6 and 12 respectively, resulted only in minor differences with the basic setting, indicating that the optimal solution is rarely restricted by these capacity bounds. For this reason, results for the larger capacity bounds are omitted.

3. **Weights**: The extension considered in Section 5.2. All ships are given a priority value which is used as a weight factor in the objective function. Lock capacity is not limited in this case. The generated instances each have ship priority values chosen randomly from $\{1, 2, 3\}$ with equal probability.

4. **Handling times**: The extension considered in Section 5.3. All ships are given a handling time. For any lock movement the lockage time is increased by the sum of the handling times of all ships present in the lock. For the instances below, handling time values are chosen uniformly random from $\{0, 2, 5\}$. Ship priorities and lock capacity are not considered. Note that finding the optimum solution is NP-hard as shown in 5.3.

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

6.2 Heuristics

Let us also define the behaviour of each of the heuristics. Each of these will be applied to all problem settings where possible. Since the solution values for these heuristics depend on the initial position of the lock (whereas it does not for the exact solution), we obtain a heuristic solution for both starting positions and report the best of both solutions.

Continuous up/down (CUD): The lock will move whenever possible, resulting in a continuous up/down movement. At the start of each movement, as many ships are moved as the lock capacity (if limited) allows. This strategy uses no information whatsoever, and even ignores the number of ships present at the lock. The first lock movement is assumed to start at $t = 0$. When ship

handling times are present, the lockage duration is adjusted accordingly. Ship waiting time is weighted by its priority value if required.

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

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

The heuristics above use no information about future arrivals. On real-world waterways however, ships should typically announce their arrival at a lock some time in advance. For example, the River Information Services being implemented in the European Union by the Central Commission for Navigation on the Rhine (2015) specifies that lock operators should be notified at least two hours before arrival. This standard already applies to all cargo ships on the Rhine and Danube rivers. More information can thus be expected to be available to the lockmaster. We present two additional heuristics below that incorporate some of this information in the decision process.

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

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

Look-ahead 2T (LA2T): Like the previous heuristic, this procedure also makes use of future arrival information. The LA2T heuristic again decides whether or not to move the lock at certain decision times. However, we now base this decision on the optimum solution for the look-ahead interval. We assume that all arrival information is known for the next $2T$ time units and we create a new subproblem over this $2T$ horizon, also including all unhandled ships that are waiting at the decision time. This subproblem is then solved with the exact solution procedure outlined in sections 3 and 5 while ensuring that the initial position of the lock is fixed. If the lock moves immediately in the solution to the subproblem, the heuristic now decides to move immediately in the original problem. If not, the lock waits until the next decision point, which corresponds to the arrival of a ship at the lock's position, or whenever a new ship is added to the $2T$ look-ahead period. Lock capacity as well as ship priorities are easy to add. Handling times cannot be included since an efficient procedure to find an exact solution is not available, as argued in 5.3. The look-ahead interval can be chosen arbitrarily large to improve the performance of this heuristic. However, repeated application of the exact procedure quickly increases the LA2T computation time beyond that of the other heuristics. For small look-ahead intervals such as $2T$, the exact procedure does find a solution quickly since the solution graph has very few edges in general.

For those problem setting where the first-come first-served assumption (i.e. property 4) does not hold, each of the heuristics requires a sequencing rule to decide on the order in which the ships are handled. This is the case for the setting where capacity, priority, and handling times are considered simultaneously. We apply a greedy strategy in this case. Intuitively, one expects that ships with high priority and/or short handling times should be handled first. A decision rule could be to order waiting ships by decreasing value of P/HT, where P is the priority value and HT the handling time. For our set of instances however, it turns out that handling ships by decreasing priority leads to significantly better performance. Many other greedy policies can be used, but are not considered

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

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

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

Table 2: Results for different problem settings. Each value represents the average of 25 instances simulating the same arrival process. The arrival parameter equals 1/15.

below. For the setting with all extensions, we report results for the decreasing priority policy.

6.3 Results

We summarize the results of all simulations in tables 1, 2, and 3. Each table shows values obtained by averaging over 25 instances for each heuristic under different problem settings and for a specific arrival probability (see 6.1). We do not separately list the required computation time as minimizing the solution time was not our main priority. Nevertheless, the exact solution is obtained in less than a second on average, except for the bounded capacity setting, where the increased graph complexity increases the computation time up to a 10 minutes average on a 3.4 GHz machine with 4GB RAM. Exact results are reported in minutes of total waiting time. Heuristic values are shown as a percentage relative to the corresponding exact solution where possible, and as total waiting time in minutes where the exact solution is not available. All generated instances and results for each individual instance are available online (`https://perswww.kuleuven.be/~u0086328/lockmasterdata.html`).

We see from the tables that the straightforward heuristics (CUD, MA, WUT) perform significantly worse than their look-ahead counterparts, though performance improves as the arrival probability increases. The WUT heuristic

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

Table 3: Results for different problem settings. Each value represents the average of 25 instances simulating the same arrival process. The arrival parameter equals 1/10.

only realizes very small improvements over MA in some cases; we expect that there may in fact be certain conditions when such a threshold policy is beneficial. Especially when capacity is considered limited, moving a lock while it is not 'sufficiently' full may introduce unnecessary waiting times in future lock movements.

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

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

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

Table 4: Exact solution and heuristic performance for the 'Poisson' instances for the basic problem setting with unbounded lock capacity.

6.4 Results for the instances by Verstichel and Vanden Berghe

Results for the instances maintained by Verstichel & Vanden Berghe (2009) are presented in Tables 4-7. Relative weights and handling times are not included in these datasets. A subset of the instances has interarrival times which follow a Poisson distribution. Results for these instances are reported in Tables 4 and 6. Other instances have times between arrivals following a uniform distribution. Results for these instances are given in Tables 5 and 7. We do not report averages for the values in the tables as each of the instances simulates an arrival process with different parameters. We refer to the original paper by Verstichel & Vanden Berghe (2009) for details on the instance parameters. For Tables 6 and 7, where capacity is considered, we ignore the ship size and assume a capacity bound equal to 3 ships. The lockage time T is chosen equal to 30.

7 Conclusion

This paper introduced the lockmaster's problem, a new problem that is closely linked to batch scheduling problems. The problem can be solved in polynomial time; we were able to build a graph such that applying a basic shortest path

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

Table 5: Exact solution and heuristic performance for the 'Uniform' instances for the basic problem setting with unbounded lock capacity.

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

Table 6: Exact solution and heuristic performance for the 'Poisson' instances for the limited capacity setting with bound equal to 3.

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

Table 7: Exact solution and heuristic performance for the 'Uniform' instances for the limited capacity setting with bound equal to 3.

algorithm solves the lockmaster's problem, and several extensions, to optimality. Computational experiments confirm that this exact algorithm outperforms a number of basic heuristics.

8 Further research

In this work we study the lockmaster's problem for a single lock. A relevant question is how to deal with the problem in case of multiple locks in series, either with ships sailing downstream at the first lock and upstream at the last lock, or, more complex, with ships also arriving at intermediate locks. In general, more complex waterway networks with several locks would be a nice subject for future work. In reality, lockmasters do not know all the ship arrival times in advance, except for ships that are already within a certain distance of the lock. Studying the online version of the lockmaster's problem could capture this element in a better way. A different direction for future research is to go further into batch scheduling with three or more job families, instead of the two families related to the lockmaster's problem studied in this paper.

Acknowledgements

This research has been partially funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office. This paper has grown out of an extended abstract (Coene & Spieksma, 2011) presented at the Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems. We would like to thank Cor Hurkens for an interesting discussion

on the subject, Rolf Möhring for pointing out references Du & Yu (2003) and Luy (2010), and Wenchao Wei for roughly translating Du & Yu (2003). We are also grateful to Rijkswaterstaat for providing real-world arrival data for the Terneuzen locks.

References

Aggarwal, A., & Park, J. K. (1993). Improved algorithms for economic lot size problems. *Operations Research*, *41*, 549–571.

Antwerp Port Authority (2011). Port of Antwerp starts building the largest lock in the world. `http://www.deurganckdoksluis.be/en/press/port-antwerp-starts-building-largest-lock-world`. (accessed October 7, 2011).

Baptiste, P. (2000). Batching identical jobs. *Mathematical Methods of Operations Research*, *52*, 355–367.

Boudhar, M. (2003). Scheduling a batch processing machine with bipartite compatibility graphs. *Mathematical Methods of Operations Research*, *57*, 513–527.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., & van de Velde, S. L. (1998). Scheduling a batching machine. *Journal of Scheduling*, *1*, 31–54.

Central Commission for Navigation on the Rhine (2015). Electronic ship reporting in inland navigation. Leaflet. URL: `http://www.ccr-zkr.org/files/documents/ris/leaferi2015_e.pdf`.

Cheng, T. C. E., Yuan, J. J., & Yang, A. F. (2005). Scheduling a batch-processing machine subject to precedence constraints, release dates and identical processing times. *Computers and Operations Research*, *32*, 849–859.

ChinaDaily (2011). Three gorges ship lock marks 8 years of operation. `http://usa.chinadaily.com.cn/china/2011-06/18/content_12730491.htm`. (accessed October 7, 2011).

Coene, S., & Spieksma, F. C. R. (2011). The Lockmaster's problem. In A. Caprara, & S. Kontogiannis (Eds.), *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems* (pp. 27–37). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik volume 20. doi:`10.4230/OASIcs.ATMOS.2011.27`.

Condotta, A., Knust, S., & Shakhlevich, N. V. (2010). Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, *13*, 463–477.

Du, J. N., & Yu, S. M. (2003). Dynamic programming model and algorithm of shiplock scheduling problem. *Computer and Digital Engineering*, *31*, 47–50. (in Chinese).

European Commission (2015). Promotion of inland waterway transport. http://ec.europa.eu/transport/inland/promotion/promotion-en.htm. (accessed 21 November 2015).

Federgruen, A., & Tzur, M. (1991). A simple forward algorithm to solve dynamic lot sizing models with n periods in o(n log n) or o(n) time. *Management Science*, *37*, 909–925.

Finke, G., Jost, V., Queyranne, M., & Sebő, A. (2008). Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, *156*, 556–568.

Goodban Belt LLC (2010). *New York State Canal System*. Technical Report. URL: `http://www.canals.ny.gov/business/modern-freightway.pdf`.

Griffiths, J. D. (1995). Queueing at the suez canal. *Journal of the Operational Research Society*, *46*, 1299–1309.

Günther, E., Lübbecke, M. E., & Möhring, R. H. (2011). Challenges in scheduling when planning the ship traffic on the kiel canal. Extended abstract, 10th Workshop on Models and Algorithms for Planning and Scheduling Problems.

Hermans, J. (2014). Optimization of inland shipping - a polynomial time algorithm for the single ship single lock optimization problem. *Journal of Scheduling*, *17*, 305–319.

Inland Navigation Europe (2014). *Annual Report*. Technical Report. URL: `http://www.inlandnavigation.eu/what-we-do/downloads/`.

Lee, C., Uzsoy, R., & Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, *40*, 764–775.

Luy, M. (2010). *Algoritmen zum Scheduling von Schleusungsvorgängen am Beispiel des Nord-Ostsee-Kanals*. Master's thesis TU Berlin. (in German).

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

Ng, C. T., Cheng, T. C. E., Yuan, J. J., & Liu, Z. H. (2003). On the single machine serial batching scheduling problem to minimize total completion time with precedence constraints, release dates and identical processing times. *Operations Research Letters*, *31*, 323–326.

Panama Canal Authority (2006). *Proposal for the expansion of the Panama Canal*. Technical Report. URL: `http://www.pancanal.com/eng/plan/documentos/propuesta/acp-expansion-proposal.pdf`.

Petersen, E. R., & Taylor, A. J. (1988). An optimal scheduling system for the Welland Canal. *Transportation Science*, *22*, 173–185.

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, *120*, 228–249.

Rijkswaterstaat (2010). Twentekanalen: uitbreiding sluis eefde. `http://www.rijkswaterstaat.nl/water/projectenoverzicht/twentekanalen-uitbreiding-sluis-eefde/`. (in Dutch, accessed October 7, 2011).

Sack, J. R., & Urrutia, J. (Eds.) (2000). *Handbook of Computational Geometry*. Amsterdam, The Netherlands: North-Holland - Elsevier.

Smith, L. D., Nauss, R. M., Mattfeld, D. C., Li, J., Ehmke, J. F., & Reindl, M. (2011). Scheduling operations at system choke points with sequence-dependent delays and processing times. *Transportation Research Part E*, *47*, 669–680.

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

Ting, C., & Schonfeld, P. (2001). Control alternatives at a waterway lock. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, *127*, 89–96.

U.S. Army Corps of Engineers (2009). *Waterborne Commerce from the United States*. Technical Report. URL: `http://www.ndc.iwr.usace.army.mil/wcsc/pdf/wcusnatl09.pdf`.

van Haastert, M. (2003). *Planningsmodel scheepvaartafhandeling bij de Noordersluis in* IJ*muiden*. Master's thesis TU Delft. (in Dutch).

Verstichel, J., De Causmaecker, P., Spieksma, F. C. R., & Vanden Berghe, G. (2014a). Exact and heuristic methods for placing ships in locks. *European Journal of Operational Research*, *235*.

Verstichel, J., De Causmaecker, P., Spieksma, F. C. R., & Vanden Berghe, G. (2014b). The generalized lock scheduling problem: An exact approach. *Transportation Research Part E*, *65*.

Verstichel, J., & Vanden Berghe, G. (2009). A late acceptance algorithm for the lock scheduling problem. *Logistik Management*, *5*, 457–478.

Wagelmans, A., Van Hoesel, S., & Kolen, A. (1992). Economic lot sizing: An o(n log n) algorithm that runs in linear time in the wagner-whitin case. *Operations Research*, *40*, S145–S156.

Webster, S., & Baker, K. R. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, *43*, 692–703.

FACULTY OF ECONOMICS AND BUSINESS
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be