No-wait scheduling for locks

Passchyn W, Briskorn D, Spieksma F.

KBI_1605

No-wait scheduling for locks

Ward Passchyn[*] Dirk Briskorn[†] Frits C.R. Spieksma[*]

March 1, 2016

Abstract

We investigate a problem inspired by the practical setting of scheduling a lock with parallel chambers. We show how this problem relates to known interval scheduling problems, as well as to a particular graph coloring problem on multiple unit interval graphs. We explore the relationships between these problems and discuss the complexity of different problem variants. In particular, for a lock consisting of two chambers we are able to characterize the feasible instances and use this result to obtain efficient algorithms. We also provide an efficient algorithm for the special case with identical lock chambers. Furthermore, we describe a dynamic programming approach for the more general case with arbitrary chambers, and prove that the problem is strongly NP-complete when the number of chambers is part of the input.

Keywords: *Lock scheduling, Graph coloring, Interval scheduling*

1 Introduction and existing literature

Locks are a necessity on many inland waterways; they maintain the water level while allowing ships traversing these waterways to overcome the resulting water level differences. We consider the following scheduling problem that deals with ships passing through a lock: consider a single lock that consists of m parallel chambers. The chambers operate independently of each other, and are each characterized by two numbers: their lockage time (denoted by T_j, $j = 1, \ldots, m$), and their capacity (denoted by C_j, $j = 1, \ldots, m$). A *lockage* or *lock movement* refers to a single operation of the lock, i.e. allowing a ship to enter and subsequently to change the water level in the lock from the downstream water level to the upstream water level, or vice versa, allowing the ship to exit the lock and continue its journey. The term *lockage time* refers to the time needed to

[*]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

e-mail addresses: {ward.passchyn, frits.spieksma}@kuleuven.be; briskorn@uni-wuppertal.de

complete this operation. The *capacity* of a lock refers to the number of ships that can be simultaneously served during a single lockage. Ships arrive at the locks at given times $t_1, t_2, \ldots, t_n$. We use $\mathcal{T}$ to denote the set of arrival times, i.e. $\mathcal{T} = \{t \mid t = t_i, 1 \leq i \leq n\}$. A ship can arrive either from the upstream side of the lock, or from the downstream side of the lock. We will refer to ships by their direction of travel, i.e. ships arriving on the upstream (downstream) side of the lock, are referred to as downstream traveling (upstream traveling) ships. Our interest in this paper is exclusively on the existence of so-called *no-wait* schedules. A no-wait schedule is a schedule where each ship, upon its arrival, can enter a chamber of the lock immediately. Thus, in a no-wait schedule, each ship i $(1 \leq i \leq n)$ leaves the lock at either $t_i + T_1$ or $t_i + T_2$ or $\ldots$ or $t_i + T_m$, depending on the particular chamber the ship is assigned to. The question we address is thus: given the arrival time and the direction of travel for each ship, does there exist an assignment of each ship to a chamber such that no ship has to wait; more compactly: does there exist a no-wait schedule?

We are aware that, from a practical point of view, this problem description does not include all relevant features such as the size of a ship and ship-dependent lockage time. However, in order to be able to solve problems, it is good to understand the behavior of this more basic problem. It is also justified to focus on the no-wait setting initially: in a number of ports, large vessels may be subject to a 'tidal window', i.e. a limited time interval where they are able to enter a port. For this reason, a policy can be enforced that sea-going vessels do not incur any waiting time at the locks that allow entry into the port area. A similar argument can be made for locks on inland waterways when serving certain classes of ships carrying dangerous cargo. Locks must then be scheduled such that these ships are immediately served by the lock upon their time of arrival.

Moreover, as we argue below, the problem can be seen as a particular interval scheduling problem, which is of independent interest, and some of our results have implications for other interval scheduling problems. A special case of the problem that is of interest is the case where all ships travel in the same direction, to which we refer as the *uni-directional* case.

1.1 Practical motivation

Scheduling locks is a problem that is receiving an increasing amount of attention. In particular, when confronted with a series of locks (e.g. along a canal), the problem of operating the locks jointly to minimize total waiting time or emissions is dealt with in Prandtstetter et al. (2015) and Passchyn et al. (2016); see also Disser et al. (2015) for a related more abstract setting.

Also, the problem of scheduling a lock consisting of one chamber is treated by Hermans (2014), who presents a $O(n^4 \log n)$ dynamic programming algorithm that asserts feasibility with respect to given ship deadlines when the single chamber has unit capacity. Passchyn et al. (2015) deal with minimizing total waiting time for a single lock chamber. They give an $O(n^4)$ algorithm for the bi-directional single chamber setting and discuss results for such practical features as ship handling times, drought, etc. Smith et al. (2011) view the single-lock single-chamber setting as a two-stage queue and describe a mixed integer programming model as well as a heuristic solution procedure.

The research mentioned above concentrates on single-chamber locks. In practice, however, many locks consist of more than one chamber. On the Panama Canal, for example, each lock consists of two identical parallel chambers. Another example is the Wijnegem lock, situated in Belgium, which connects the Albert Canal to the port of Antwerp. Like all other locks on the Albert Canal, this lock consists of three non-identical chambers. Furthermore, the construction of a fourth lock chamber in Wijnegem is currently under consideration, see also Waterwegen en Zeekanaal NV and nv De Scheepvaart (2014).

Ting and Schonfeld (2001) mention a heuristic for a lock consisting of two chambers. The only work we are aware of that deals with exact methods for scheduling a lock with multiple chambers is Verstichel et al. (2014); a mixed integer program is proposed that simultaneously decides upon the packing of ships in chambers, and the operating times of the chambers, see also Verstichel (2013). Thus, studying a single lock that consists of multiple chambers is a practical, largely unexplored problem.

1.2 Interval scheduling

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

Interval scheduling is a well-studied subject, see Kolen et al. (2007) for an overview. A recent paper by Krumke et al. (2011) deals with interval scheduling on related machines: given are m machines, each with a certain speed s_j ($1 \leq j \leq m$), and n intervals specified by a starting point r_i and a processing time p_i ($1 \leq i \leq n$). They show that even deciding the existence of a schedule is NP-complete. This setting is related to the uni-directional variant of our problem: by setting the speed of chamber j equal to $T_j/T_{\max}$ (where $T_{\max}$ refers to the maximum lockage time over the chambers, i.e. $T_{\max} = \max_j T_j$), our problem is seen to be a special case of the problem in Krumke et al. (2011) since in our case the lengths of intervals corresponding to a particular machine are identical.

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

1.3 Graph coloring

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

The two problem descriptions are related as follows. Consider the uni-directional case of the problem formulated in the "lock"-description and assume that all arrival times are distinct. Build a node set V by having a node in V for each arrival of a ship. Next, build an edge set E_j by having an edge in E_j between the nodes $i_1, i_2 \in V$ if and only if $t_{i_1} + T_j > t_{i_2}$ for $j = 1, \ldots, m$. Observe that if the lock chambers are ordered such that $T_1 \leq \ldots \leq T_m$, this immediately implies that $E_1 \subseteq \ldots \subseteq E_m$. The existence of a partition of V into subsets V_j, with V_j an independent set in (V, E_j) for all $j \in \{1, \ldots, m\}$, then corresponds to the existence of a no-wait schedule. Indeed, the nodes, or arrivals, in V_j are handled by chamber j in a corresponding solution to the "lock" problem.

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

1.4 Summary of results

We use the "lock" description of the problem to describe our results. Thus, our results are formulated in terms of locks and ships; for example, the phrase 'assigning ship i to chamber j' corresponds to 'executing job i on machine j' in the "interval" description, and to 'placing node i in subset V_j' in the "graph" description.

As mentioned before, the input of our problem consists of lockage times T_j and capacities C_j for $j \in \{1, \dots, m\}$, and arrival times t_i and direction $d_i \in \{\text{upstream}, \text{downstream}\}$ for $i \in \{1, \dots, n\}$. We consider the question "does there exist a no-wait solution?". We refer to this problem as "no-wait lock scheduling" (NLS). In addition to this problem, we consider different special cases in what follows, deciding on whether a no-wait solution exists subject to additional restrictions imposed on the input. For convenience, we introduce a notation to refer to these special cases. We refer to the problem settings as NLS{-uni, ∅}{-2, -m, ∅}{-id, ∅}{-distinct, ∅}, where *uni* refers to the uni-directional case (omitting this implies bi-directional traffic), *2* refers to the setting with two lock chambers and m refers to the setting where the number of chambers m is fixed (omitting these implies that the number of chambers is part of the input), *id* refers to the setting with identical chambers (omitting this implies that values for lockage time and capacity are arbitrary), and *distinct* refers to the setting where all arrival times are distinct (omitting this implies that multiple ships may arrive simultaneously). We point out that specifying additional parameters in the problem name increasingly yields a more specific case of the problem. NLS, which describes the setting with an arbitrary number of non-identical chambers and bi-directional traffic, thus describes the most general problem covered by this notation. Table 1 lists the notation, as well as the results discussed in this paper, for the different special cases of the problem that we consider. Based on the chamber characteristics, we consider the following settings:

1. The setting where $m = 2$ with two arbitrary chambers. For the uni-directional setting, called NLS-uni-2, we give necessary and sufficient conditions for deciding on the existence of a no-wait schedule (Section 2), and we show how to find such a schedule in $O(n)$ time, provided that the arrival times are sorted (see Section 1.5). Further, for the bi-directional case NLS-2, we give a reduction to 2-SAT that leads to an $O(n^2)$ algorithm (Section 3).

2. The setting where $C_j = C$ and $T_j = T$ for all $j \in \{1, \dots, m\}$, i.e. the setting with m identical chambers. The resulting problems are called NLS-uni-id

	2 arbitrary chambers	m identical chambers	m arbitrary chambers	m arbitrary chambers (m part of the input)
uni-directional	NLS-uni-2 $O(n)^\dagger$ (Section 2)	NLS-uni-id $O(n)^\dagger$ (implied)	NLS-uni-m $O(mn^m)$ (implied)	NLS-uni strongly NP-complete (Section 6)
bi-directional	NLS-2 $O(n^2)$ (Section 3)	NLS-id $O(n)^\dagger$ (Section 4.3)	NLS-m $O(mn^m)$ (Section 5)	NLS strongly NP-complete (implied)

Table 1: Summary of results. † These results apply to input with arrival times given in sorted order, see Section 1.5.

and NLS-id, and we show that we can solve these problems in $O(n)$ time for sorted arrival times (Section 4).

3. The setting where the number of chambers m is fixed. We give a dynamic program (DP) for our problem that runs in polynomial time (Section 5). For the problem of finding a feasible no-wait schedule described here, this DP strengthens the result in Sung and Vlach (2005) that can be applied to the uni-directional case.

4. The setting with an arbitrary number of chambers. We prove that the uni-directional case of this variant is NP-complete (Section 6). This result strengthens both the result given in Krumke et al. (2011) and a result in Böhmová et al. (2013).

1.5 A note on sorted arrival times

Notice that some of the results stated in Table 1 apply exclusively to sorted input, i.e. these results require the assumption that arrival times are given in non-decreasing order. Due to the well-known $\Omega(n \log n)$ lower bound on comparison-based sorting algorithms, it may not be possible to improve beyond a complexity of $O(n \log n)$ in the general case of unsorted arrival times. However, assuming that the arrival times are known in sorted order may be justified. For instance, when iteratively applying the presented methods, a large part of the input may have remained unchanged since earlier iterations, so that sorting is no longer required for a large subset of the given arrival times. Furthermore, the input may be unsorted but may satisfy additional assumptions which allow the use of a non-comparison-based sorting algorithm such as radix sort, which runs in linear time. Therefore, we choose to report the complexity assuming that the arrival times are sorted.

2 Two arbitrary chambers, uni-directional case

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

In Section 2.1, we first look at problem NLS-uni-2-distinct, i.e. the special case where (i) all ships travel in the same direction, and (ii) all arrival times are distinct. Notice that the numbers C_1, C_2 are then irrelevant since a chamber contains at most one ship. In Sections 2.2 and 2.3, we extend our approach to a more general setting where some ships are pre-assigned to a chamber and describe how this result can be used to model the setting NLS-uni-2, where the numbers C_1 and C_2 become relevant.

2.1 Uni-directional setting with distinct arrival times

As argued in Section 1.5, we assume throughout this section that the arrival times are given in sorted order. We describe an $O(n)$ algorithm under this assumption. We organize this section as follows: in Section 2.1.1 we describe the construction of a graph corresponding to an instance of NLS-uni-2-distinct, and discuss some basic observations. In Section 2.1.2 we prove a theorem characterizing feasibility, and Section 2.1.3 describes an $O(n)$ algorithm.

2.1.1 Graph and concepts

An instance $\mathcal{I}$ is given by specifying distinct ordered arrival times $t_1 < t_2 < \ldots < t_n$ and lockage times $T_1 < T_2$. We say that an instance is feasible if there exists a no-wait solution, otherwise it is not feasible. Given an instance $\mathcal{I}$, we create a graph $G(\mathcal{I})$; we will, in the sequel, simply write G. Notice that we allow multiple edges between a pair of nodes in G; more precisely, the edge set of G consists of a set E^S and a set E^L. The graph is constructed as follows. There is a node for each arrival time; two nodes $i < j$ are connected via an *s-edge* $(i,j) \in E^S$ if and only if $t_j - t_i < 2T_1$; two nodes $i < j$ are connected via an *l-edge* $(i,j) \in E^L$ if and only if $t_j - t_i < 2T_2$. Observe that (V, E^S) and (V, E^L) are unit interval graphs by construction, and that $E^S \subseteq E^L$. In fact, deciding whether the set of nodes V can be partitioned into two subsets which are independent sets in E^S and E^L respectively corresponds exactly to the problem stated in the "graph"-description of our problem, defined in Section 1.3, for $m = 2$, where $E^S = E_1$ and $E^L = E_2$.

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

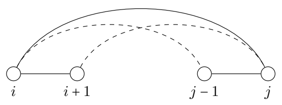

Figure 1: Structure described in Observation 2.3. The existence of $(i,j) \in E^L$ implies that $(i,j-1) \in E^L$ and $(i+1,j) \in E^L$ since $t_i < t_{i+1} < t_{j-1} < t_j$

A no-wait solution to the given instance can exist only if each ship can be assigned to one of the two lock chambers so that a lockage starts at the ship's time of arrival. We can immediately observe the following: for a solution to be no-wait, two arrivals connected by an s-edge cannot both be served by the short chamber; two arrivals connected by an l-edge cannot both be served by the long chamber. Furthermore, we can observe a number of interesting properties in graph G. In addition to highlighting some structural characteristics of an instance and its graph, we will refer to these observations in the proof that follows.

Observation 2.1. If there exists a node $i \in V \setminus \{n\}$ such that $(i,i+1) \notin E^L$, i.e. if there exists a pair of consecutive nodes which are not connected by an l-edge, the instance splits into two independent sub-problems. This is easily seen since both chambers are then available at time t_{i+1} for any no-wait solution, regardless of the assignment of nodes $1,\dots,i$. A no-wait solution thus exists if and only if there is a no-wait solution for each of the two sub-problems.

Observation 2.2. If there exists a node $i \in V$ such that $(i,i+2) \in E^S$, i.e. if there exists a triangle of s-edges in G, then the instance is not feasible. This readily follows from the fact that there does not exist a proper 2-coloring in a triangle graph.

Observation 2.3. If an l-edge contains two s-edges that are node-disjoint, i.e. if there exist nodes $i,j \in V$ with $j > i+2$ so that $(i,i+1) \in E^S$, $(j-1,j) \in E^S$, and $(i,j) \in E^L$, then the instance is not feasible. This is easily verified by enumerating all possible assignments for nodes i, $i+1$, $j-1$, and j. An example is shown in Figure 1.

Observation 2.4. If there exists a node $i \in V$ such that there exist s-edges $(i,i{+}1),(i{+}1,i{+}2),(i{+}2,i{+}3) \in E^S$ and l-edges $(i,i{+}2),(i{+}1,i{+}3) \in E^L$, the instance is not feasible. Figure 2 shows the graph for an instance consisting of 4 such nodes. It is easily verified (see also the next observation) that both nodes $i+1$ and $i+2$ must be served by the short chamber in all feasible solutions, which is impossible due to the presence of s-edge $(i+1,i+2)$. Hence, an instance containing this structure is not feasible.

For convenience, we restrict ourselves in the remainder of Section 2 to instances $\mathcal{I}$ whose corresponding graphs $G(\mathcal{I})$ do not contain any of the structures

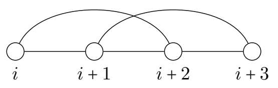

Figure 2: Structure described in Observation 2.4.

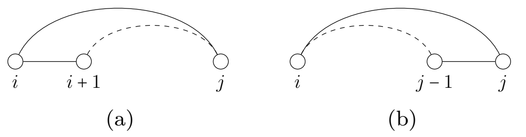

Figure 3: Structures described in Observation 2.5. Note that the existence of the dashed edges is implied.

described in Observations 2.1 to 2.4. This can be done without loss of generality, since we can recognize these structures efficiently, see Section 2.1.3. Furthermore, an additional observation allows us to recognize certain nodes which must be assigned to the short chamber:

Observation 2.5. If there exist nodes $i, j \in V$ with $j > i+1$ such that $(i, i+1) \in E^S$ and $(i, j) \in E^L$, a feasible solution can exist only when node j is assigned to the short chamber. This follows readily from the fact that either i or $i+1$ must be assigned to the long chamber, and an l-edge to j starts at both i and $i+1$. From symmetry, it immediately follows that the same holds for a node i where $(i, j) \in E^L$ and $(j, j-1) \in E^S$. The associated graphs are shown in Figures 3a and 3b.

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

Definition 1. *Given an instance $\mathcal{I}$ and graph $G(\mathcal{I})$, a* bad path *is any sequence of distinct nodes $(i_1, i_2, \ldots, i_k)$ with $k \geq 4$ that satisfies:*

1. *The nodes in the sequence appear in the order defined on V, with exception of i_1 and i_k, which satisfy $i_2 < i_1 < i_3$ and $i_{k-2} < i_k < i_{k-1}$. More formally: $i_x < i_{x+1}$ for all $x \in \{2, \ldots, k-2\}$, $i_2 < i_1 < i_3$, and $i_{k-2} < i_k < i_{k-1}$.*
2. *The pairs of consecutive nodes in the sequence are alternatingly connected by an s-edge and an l-edge, with the first and last edges in the sequence being both s-edges. More formally: $(i_x, i_{x+1}) \in E^S$ for all odd $x \in \{1, \ldots, k-1\}$, $(i_x, i_{x+1}) \in E^L$ for all even $x \in \{1, \ldots, k-1\}$, and k is even.*

Note that a bad path necessarily contains an even number of nodes. An example is shown in Figure 4. Observe that a bad path with $k = 4$ is present Figure 1.

We also describe specific nodes that are closely related to the definition of a bad path. Intuitively, we define a *potentially bad node* as the latest node in a

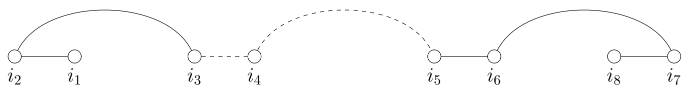

Figure 4: Example of a bad path. Note that the structure described in Definition 1 remains satisfied if the dashed edges are replaced with a longer sequence (with odd number of nodes) consisting alternatingly of s-edges and l-edges.

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

Definition 2. *A node i_k is a* potentially bad node *if there exists a path (i_1, i_2, …, i_k) that satisfies the following:*

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

Observe that in a bad path, all nodes i_j in the path with $j \geq 5$ and j odd are potentially bad nodes. E.g. in Figure 4, nodes i_5 and i_7 are potentially bad nodes, whereas i_3 is not.

2.1.2 Characterizing feasible instances

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

Theorem 1. *An instance $\mathcal{I}$ of NLS-uni-2-distinct is feasible if and only if its corresponding graph $G(\mathcal{I})$ does not contain a bad path.*

Proof. $\Rightarrow$ We argue that if $G(\mathcal{I})$ contains a bad path, the instance is not feasible. Consider a bad path $(i_1, \dots, i_k)$ in G. It follows from Observation 2.5 and $i_2 < i_1 < i_3$ that node i_3 must be assigned to the short chamber. We now trace the path from i_3 to i_{k-1}. Note that, since l-edges and s-edges alternate, any solution must assign nodes $i_3, \dots, i_{k-1}$ to the short and long chamber in alternating order. This implies that node i_{k-2} must be assigned to the long chamber, while node i_{k-1} must be assigned to the short chamber. However, the bad path implies that $(i_{k-1}, i_k) \in E^S$ and $(i_{k-2}, i_k) \in E^L$ so that no chamber is available for i_k and hence no feasible solution exists.

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

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

Property 2.1. *For each $i \in V^*$, there exists a feasible solution in $G^* \setminus \{i\}$ with distinct nodes $j_s, j_l \in V^* \setminus \{i\}$ such that $(i, j_s) \in E^{S*}$, $(i, j_l) \in E^{L*}$ while j_s (j_l) is assigned to the short (long) chamber in that feasible solution.*

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

Property 2.2. *G^* contains at least one potentially bad node.*

Proof. Consider the earliest node, say i_2, in G^*. Applying Property 2.1 to this node, it follows that there must exist two additional nodes i_1, i_3 such that $(i_2, i_1) \in E^{S*}$ and $(i_2, i_3) \in E^{L*}$. We select i_1, i_3 to be minimal, i.e. i_1 is the successor of i_2 and i_3 is the successor of i_1. We obtain the structure shown in Figure 3. Applying Property 2.1 to node i_3 implies that there is an s-edge incident to node i_3. We can distinguish two possible cases:

1. The s-edge is incident to a node $i_4 > i_3$. Observe that i_4 must the be the successor of i_3. We obtain the structure shown in Figure 5a. If G^* consists of four nodes, applying Property 2.1 to i_1 implies that $(i_1, i_4) \in E^{L*}$. Observe that i_3 cannot be selected as node j_l in the property, since i_3 must not be assigned to the long chamber (Observation 2.5). Furthermore, observe that $(i_1, i_3) \notin E^{S*}$ (by our assumption that the structure from Observation 2.4 is not present in an instance) and that $(i_2, i_4) \notin E^{L*}$ (by our assumption that the structure from Observation 2.3 is not present). Then, however, G^* corresponds to an instance that is feasible, which is not the case. This is illustrated in Figure 5b. Thus, G^* consists of at least five nodes. Observation 2.1 implies that $(i_4, i_5) \in E^{L*}$, and thus i_5 is a potentially bad node.

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

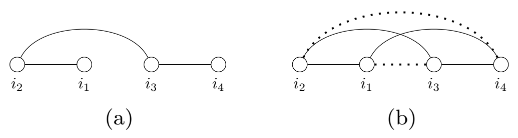

Figure 5: Structures described in case 1 in the proof. Note that, in (b), the dotted edges are not present in G^*, and a feasible solution assigns i_1 and i_3 to the short chamber and i_2 and i_4 to the long chamber.

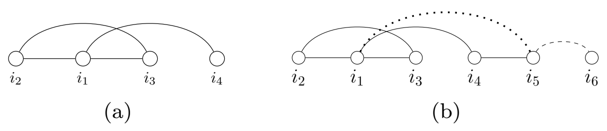

Figure 6: Structures described in case 2 in the proof.

is the successor of i_3. Applying Property 2.1 to node i_1, it follows that $(i_1, i_4) \in E^{L*}$. Observe that nodes i_2 and i_3 cannot be selected as node j_l in the property since they must not be assigned to the long chamber (Observation 2.5). G^* then contains the structure shown in Figure 6a. We then apply Property 2.1 to node i_4. Since we may assume that $(i_3, i_4) \notin E^{S*}$ (otherwise the case described above applies), it follows that there exists a node $i_5 > i_4$ with $(i_4, i_5) \in E^{S*}$. The resulting structure is shown in Figure 6b. Finally, we apply Property 2.1 again, now to i_5. It follows that there is an l-edge incident to i_5. As before, note that nodes i_3 and i_4 cannot be chosen as node j_l in the property since they must not be assigned to the long chamber (Observation 2.5). Since the existence of the l-edge (i_1, i_5), dotted in Figure 6b, implies a bad path, this l-edge cannot be present and there thus exists a node $i_6 > i_5$ with $(i_5, i_6) \in E^{L*}$. Node i_6 is then a potentially bad node, with path $(i_3, i_1, i_4, i_5, i_6)$ satisfying the definition above.

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

Property 2.2 implies that G^* contains a latest potentially bad node. We now show that this leads to a contradiction. Figure 7 illustrates the reasoning below. Let i_k be the latest potentially bad node in G^*. Applying Property 2.1 to i_k implies that there exists an s-edge (i_k, i_{k+1}). Since i_k is a potentially bad node, $i_{k+1} < i_k$ would imply the existence of a bad path. Node i_{k+1} is thus the successor of i_k. Applying Property 2.1 again, now to i_{k+1}, implies the existence of an additional l-edge. Observe that the l-edge (i_{k-1}, i_{k+1}), illustrated by the dotted l-edge in Figure 7, cannot be present because it implies the existence of a bad path. Thus, there exists a node $i_{k+2} > i_{k+1}$ with $(i_{k+1}, i_{k+2}) \in E^{L*}$. Note that i_{k+2} is a potentially bad node and $i_{k+2} > i_k$. This contradicts our choice of i_k as the latest potentially bad node.

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

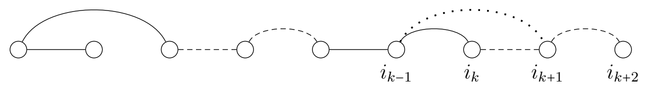

Figure 7: Structure describing the latest potentially bad node in G. Observe that the dotted l-edge cannot be present.

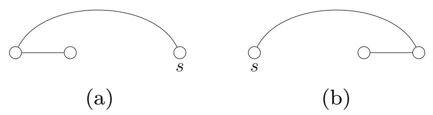

Figure 8: The nodes labeled s must be assigned to the short chamber.

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

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

Recall that in Figure 8a and Figure 8b, each of the nodes labeled s must be assigned to the short chamber in any feasible solution, as argued in Observation 2.5.

The outline of the procedure is as follows. We first identify all nodes that must be assigned to the short chamber due to the structures shown in Figure 8a and Figure 8b. We then use implications from these assignments to assign other nodes to the chambers. In this way, all 'forced' assignments are handled. Finally, we apply a simple greedy procedure to assign the remaining nodes to chambers. In the remainder of this section, we will show that if no bad paths are present, the greedy procedure always yields a feasible assignment of lock chambers (i.e. correctness of the algorithm), and that each of these steps can be performed in linear time (i.e. complexity of the algorithm).

We will call any node that must be assigned to the same chamber in all feasible solutions due to the implications mentioned above, a *fixed node*. We distinguish *s-fixed* and *l-fixed* nodes for nodes that must be assigned to the short and long chambers respectively. As argued in Observation 2.5, the nodes labeled s in Figure 8 are s-fixed nodes. We start by identifying all occurrences of these

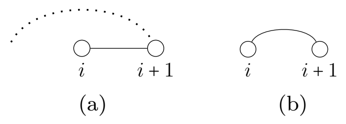

Figure 9: Assigning $i + 1$ to a chamber has no implications for nodes earlier than i.

structures in G. Observe that these structures correspond to the 'beginning' (i.e. the first three nodes) and the 'end' (i.e. the last three nodes) of a bad path. Finding these structures is easily done in linear time by considering each node once and checking for the presence of the edges shown in the figures. We obtain an initial set of s-fixed nodes. Initially, no nodes are l-fixed.

Observe that any node connected to an s-fixed node by an s-edge is necessarily l-fixed, and any node connected to an l-fixed node by an l-edge is necessarily s-fixed. The following step is to identify all remaining nodes that can be fixed using these observations. We first consider all nodes in order; let i be the current node. If i is s-fixed and $(i, i + 1) \in E^S$, add $i + 1$ to the set of l-fixed nodes. If i is l-fixed, add all $j > i$ for which $(i, j) \in E^L$ to the set of s-fixed nodes. Next, we repeat this for all nodes in reverse order; again let i be the current node. If i is s-fixed and $(i, i - 1) \in E^S$, add $i - 1$ to the set of l-fixed nodes. If i is l-fixed, add all $j < i$ for which $(i, j) \in E^L$ to the set of s-fixed nodes. Clearly, when a node is both s-fixed and l-fixed, no feasible solution exists. Observe that such a conflict is only possible if a path starting from the 'beginning' of a bad path is extended to a node where it meets the 'end' of a bad path.

It is important to note that nodes fixed by considering all nodes in order do not in turn fix any earlier nodes. Similarly, nodes fixed by considering all nodes in reverse order have no impact on later nodes. This is illustrated in Figure 9. In Figure 9a, assume that i is an s-fixed node. Adding $i + 1$ to the set of l-fixed nodes then has no new implications for any nodes earlier than i because if an l-edge $(j, i + 1)$ exists with $j < i$, the initial set of s-fixed nodes already contained node j. In the rightmost figure, assume that i is an l-fixed node. Adding $i + 1$ to the set of s-fixed nodes then has no implications for any nodes earlier than i since the existence of an s-edge $(j, i + 1)$ with $j < i$ implies a triangle of s-edges, which is not present due to Observation 2.2. The similar statement where nodes are considered in reverse order follows immediately from symmetry.

It follows that after considering all nodes once in order and once in reverse order, no new nodes can be fixed. As noted above, if a bad path exists, at least one node must be both s-fixed and l-fixed. If no such contradiction is found, the instance is thus feasible. In the corresponding solution, all s-fixed nodes are assigned to the short chamber and all l-fixed nodes are assigned to the long chamber. Upon completing this step, there may remain nodes with no chamber assignment. Note that none of the fixed nodes has any implications for any of the remaining nodes, since this would imply that additional nodes should be fixed nodes. We describe a straightforward greedy rule that assigns the remaining nodes to the chambers and show that applying this rule to a node does not

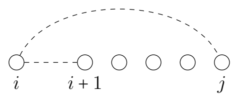

Figure 10: Illustration of the assignment rule for remaining nodes. Note that only one of the dashed edges may be present for any $j > i+1$.

prevent us from applying it to any later node, thus yielding a feasible solution. The greedy procedure considers all nodes in order. Let i be the current node to be labeled: if $(i-1,i) \in E^S$ and $i-1$ is assigned to the short chamber, assign i to the long chamber; otherwise assign i to the short chamber. To see that applying this rule does not prevent us from applying it to any later node, consider node i in Figure 10. If $(i,i+1) \in E^S$, then node $i+1$ must be assigned to the long chamber only if i is assigned to the short chamber; this is consistent with the rule. If there exists a $j > i+1$ for which $(i,j) \in E^L$, no s-edges may be incident to any nodes k with $i < k < j$ because this would imply that either i or j is a fixed node and hence is already assigned to a chamber. All such nodes k, as well as node j can thus be assigned to the short chamber, which is consistent with the rule. Note that it cannot be the case that both $(i,i+1) \in E^S$ and there exists a $j > i+1$ with $(i,j) \in E^L$ since this would imply that j is a fixed node.

After applying the greedy rule, all nodes have been assigned to a chamber; if no bad path was identified, we have thus obtained a feasible solution. A pseudo-code of this procedure is presented in Algorithm 1. Note that the algorithm assumes that $n \geq 3$; instances with $n < 3$ can be solved trivially. The discussion above establishes correctness of the algorithm.

It remains to show that the procedure described above runs in linear time. As argued above, finding the initial set of s-fixed nodes takes at most $O(n)$ time. Finding all additional s-fixed nodes is also possible in linear time. Indeed, each node has at most two s-edges incident to it and each node is considered only once in order, and once in reverse order. Graph G contains up to $O(n^2)$ l-edges; to see that finding all additional l-fixed nodes takes only $O(n)$ time, we make use of the fact that (V, E^L) is an interval graph. By definition, it follows that if l-edge (u,v) exists, all l-edges (u,w) with $u < w < v$ must also exist. Thus, when traversing the nodes in order, if a node was labeled as s-fixed due to the presence of an l-edge, no nodes earlier than this fixed node need to be checked at a later time, since such nodes are already s-fixed. Thus, it suffices to remember the latest node that was s-fixed to avoid checking all possible l-edges for each of the nodes. In Algorithm 1, this is achieved by keeping track of j, which represents the next node to be checked for the existence of an l-edge. The same argument applies when considering nodes in reverse order, where it suffices to start from the earliest s-fixed node. Since in each iteration either i or j increases and no nodes earlier than j are checked, finding the s-fixed nodes completes in $O(n)$ time.

Finally, we note that recognizing the structures described in Observations 2.1 to 2.4, which were assumed not to be present in the graph, can be achieved in

```
// input: arrival times $t_1 < t_2 < \ldots < t_n$ and lockage durations $T_1 < T_2$
$v_i \leftarrow$ node corresponding to the arrival at time $t_i$, for all $i \in \{1, \ldots, n\}$
s-fixed $\leftarrow \emptyset$
l-fixed $\leftarrow \emptyset$
// identify initial s-fixed nodes:
for $i = 1$ to $|V|$ do
    $j \leftarrow \max(i + 2, j)$
    if $(v_i, v_{i+1}) \in E^S$ then
        while $(v_i, v_j) \in E^L$ do
            s-fixed $\leftarrow$ s-fixed $\cup \{v_j\}$
            $j \leftarrow j + 1$
$j = |V|$
for $i = |V|$ to $1$ do
    $j \leftarrow \min(i - 2, j)$
    if $(v_{i-1}, v_i) \in E^S$ then
        while $(v_j, v_i) \in E^L$ do
            s-fixed $\leftarrow$ s-fixed $\cup \{v_j\}$
            $j \leftarrow j - 1$
// extend implications of fixed nodes:
for $i = 1$ to $|V|$ do
    if $v_i \in$ s-fixed and $(v_i, v_{i+1}) \in E^S$ then l-fixed $\leftarrow$ l-fixed $\cup \{v_{i+1}\}$
    $j \leftarrow i + 2$
    if $v_i \in$ l-fixed then
        while $(v_i, v_j) \in E^L$ do
            s-fixed $\leftarrow$ s-fixed $\cup \{v_j\}$
            $j \leftarrow j + 1$
for $i = |V|$ to $1$ do
    if $v_i \in$ s-fixed and $(v_{i-1}, v_i) \in E^S$ then l-fixed $\leftarrow$ l-fixed $\cup \{v_{i-1}\}$
    $j \leftarrow i - 2$
    if $v_i \in$ l-fixed then
        while $(v_j, v_i) \in E^L$ do
            s-fixed $\leftarrow$ s-fixed $\cup \{v_j\}$
            $j \leftarrow j - 1$
// check for conflicts:
if s-fixed $\cap$ l-fixed $\neq \emptyset$ then return 'not feasible'
// assign nodes to chambers using a greedy rule for non-fixed nodes:
for $i = 1$ to $|V|$ do
    if $v_i \in$ s-fixed then chambers$_i \leftarrow$ 'short'
    else if $v_i \in$ l-fixed then chambers$_i \leftarrow$ 'long'
    else if $(v_{i-1}, v_i) \in E^S$ and chambers$_{i-1}$ = 'short' then chambers$_i \leftarrow$ 'long'
    else chambers$_i \leftarrow$ 'short'
return chambers
```

Algorithm 1: Pseudo-code for the uni-directional problem with two arbitrary chambers and distinct arrival times. Note: $n \geq 3$ is assumed for the input.

linear time. This is easily seen for Observations 2.1, 2.2 and 2.4, which deal only with adjacent nodes. To recognize these structures, it suffices to traverse each of the nodes and check for the existence of the incident edges described in the observations. For Observation 2.3, we use the interval graph structure in the same was as when recognizing the s-fixed nodes in Algorithm 1. That is, we remember the latest node that was s-fixed in order to avoid checking earlier nodes which are already known to be s-fixed. If any of these structures is identified, it immediately follows that no feasible solution exists.

2.2 Fixed chamber assignments

We describe here how the algorithm can be extended to the case where some nodes are pre-assigned to a specific chamber. Note that when specific chamber assignments are imposed, an instance may not have a feasible solution while its corresponding graph does not contain a bad path. Our algorithm, however, is able to take these given assignments into account.

If a given subset of the nodes, say $S \subseteq V$, is pre-assigned to the short chamber, we initialize, in Algorithm 1, the set of s-fixed nodes to this set S. Similarly, if a given subset of nodes, say $L \subseteq V$, is pre-assigned to the long chamber, we initialize, in Algorithm 1, the set of l-fixed nodes to this set L. Clearly, when a node is both s-fixed and l-fixed, the instance is not feasible. Once the sets of fixed nodes are initialized, the initial sets of fixed nodes are extended as in Algorithm 1. The analysis of the assignment rule for all nodes that are not fixed remains valid. Note that feasibility is still identified by verifying for each node whether it is both s-fixed and l-fixed. These adjustments suffice to solve the generalization with fixed chamber assignments. The computational complexity remains unchanged.

2.3 Simultaneous arrivals

In this section, we will consider the generalization where simultaneous arrivals can occur, i.e. where arrival times need not be distinct. Since a chamber may then simultaneously serve more than one ship in a no-wait solution, we need to take the capacity of the chambers into account. Let $C_{\text{small}} = \min(C_1, C_2)$ and $C_{\text{large}} = \max(C_1, C_2)$. Note that C_{small} does not necessarily correspond to the chamber with the shortest lockage duration. We now show how to modify graph G and Algorithm 1 in order to determine whether a feasible solution exists in this setting with simultaneous arrivals.

For each time $t \in \mathcal{T}$, let k_t be the number of ships arriving at time t. Clearly, if there exists a t with $k_t > C_{\text{small}} + C_{\text{large}}$, the instance is not feasible since the number of arriving ships at time t exceeds the combined capacity of both chambers. Let us thus assume that the instance has $k_t \leq C_{\text{small}} + C_{\text{large}}$ for all $t \in \mathcal{T}$. For each $t \in \mathcal{T}$, we distinguish three cases:

1. Case 1: $2 \leq k_t \leq C_{\text{small}}$. We modify graph G as follow: we let a single node represent all the simultaneous arrivals at time t. Either chamber is a valid

assignment to simultaneously serve all k_t ships; we may thus treat these simultaneous arrivals as a single ship.

2. Case 2: $C_{\text{small}} < k_t \leq C_{\text{large}}$. As in the case above, let a single node represent all simultaneous arrivals at time t. In addition, we impose that this node must be assigned to the large chamber. Recall that the large chamber may be either the short or the long chamber, depending on the values of T_1, T_2, C_1, C_2. It is easily seen that it is never required to use both chambers to serve these k_t ships, since the the large chamber must be used regardless, and this chamber suffices to serve all ships simultaneously.

3. Case 3: $C_{\text{large}} < k_t \leq C_{\text{small}} + C_{\text{large}}$. Again, let a single node represent all simultaneous arrivals at time t. It follows that since $k_t > C_{\text{large}}$, both chambers must be used simultaneously to transfer all ships without introducing waiting time. We add this node to the set B designated to identify all nodes that must be assigned to both chambers. We argue below that, after modifying the graph for all $t \in \mathcal{T}$, the algorithm is easily adjusted to enforce this assignment for each node in B.

After graph G has been modified, we take B into account by initializing the algorithm as follows. For each node $i \in B$, we add all implications that follow from imposing that i is assigned to both chambers: for all $j \in V$ with $(i,j) \in E^S$, add j to the set l-fixed; for all $j \in V$ with $(i,j) \in E^L$, add j to the set s-fixed. It is easily argued that each of the added implications must hold in any feasible solution. When assigning the nodes to chambers, we set chambers$_i$ = 'short + long' for all $i \in B$. Furthermore, for a feasible solution to exist, it must hold that none of the nodes in B are fixed when running the algorithm. Indeed, if a node $k \in B$ with corresponding time t_k would be fixed, this implies that at least one chamber is unavailable at time t_k, so that there is no feasible assignment for k. In addition to modifying the initialization of sets s-fixed and l-fixed, we thus extend the check for conflicts in the algorithm to "if s-fixed $\cap$ l-fixed $\neq \emptyset$ or s-fixed $\cap$ $B \neq \emptyset$ or l-fixed $\cap$ $B \neq \emptyset$".

Using the result for fixed chamber assignments described in Section 2.2, and by modifying Algorithm 1 as outlined above, it follows that we can solve the resulting instance in $O(n)$ time. The following theorem concludes this discussion.

Theorem 2. *If a no-wait solution exists for the uni-directional lock scheduling problem with two chambers it can be found, i.e. problem NLS-uni-2 can be solved, in $O(n)$ time.*

3 Two arbitrary chambers

We now focus on the more general setting with two lock chambers, where ships may travel in both directions. Böhmová et al. (2013) mention a reduction to 2-SAT for an interval scheduling problem which generalizes the uni-directional setting. The well-known 2-SAT problem can be solved in a number of operations which is linear in the number of clauses, see Even et al. (1976) and Aspvall et al.

(1979). We show here that the same applies to the bi-directional two-chamber setting and describe the reduction explicitly.

Lemma 1. *An instance of NLS-2 can be modeled as an instance of 2-SAT using $O(n)$ variables and $O(n^2)$ clauses.*

Proof. In the NLS-2 setting ships may arrive simultaneously. To take this into account, we first describe how each instance of NLS-2 can be transformed into an equivalent instance where $C_1 = C_2 = 1$ and where some ships are pre-assigned to chambers. We then provide a reduction to 2-SAT for the setting with two unit-capacity chambers and pre-assigned ships.

Similar to the approach discussed in Section 2.3, we distinguish multiple cases when constructing the instance with unit capacity. For each time $t \in \mathcal{T}$, let $k_{t,d}$ be the number of ships arriving at time t and traveling in direction d. Clearly, the instance is not feasible if there exist a t and d such that $k_{t,d} > C_1 + C_2$. Since this can be easily verified, we assume that $k_{t,d} \le C_1 + C_2$ for all $t \in \mathcal{T}$, $d \in \{\text{upstream, downstream}\}$.

Consider a given instance $\mathcal{I}$ of NLS-2. As in Section 2.3, let $C_{\text{small}} = \min(C_1, C_2)$ and $C_{\text{large}} = \max(C_1, C_2)$. We construct an instance $\mathcal{I}_{\text{unit}}$, where $C_1^{\mathcal{I}_{\text{unit}}} = C_2^{\mathcal{I}_{\text{unit}}} = 1$, leaving the lockage times and the set of arrival times unaltered. The set of arriving ships is constructed as follows. For each $t \in \mathcal{T}$ and $d \in \{\text{upstream, downstream}\}$:

1. If $k_{t,d} \le C_{\text{small}}$, replace these $k_{t,d}$ ships by a single ship traveling in direction d, arriving at time t. It is immediately clear that either lock chamber suffices to handle all $k_{d,t}$ ships, so that we effectively ignore these simultaneous arrivals.

2. If $C_{\text{small}} < k_{t,d} \le C_{\text{large}}$, replace all these ships by a single ship traveling in direction d, arriving at time t, and additionally impose that this ship must be assigned to the large chamber (i.e. the second chamber if $C_1 \le C_2$, the first chamber otherwise). To serve all ships without introducing waiting time, the large chamber must be used to serve at least one ship arriving at time t and traveling in direction d. By pre-assigning the ship to the large chamber, it follows that this chamber must be available at time t to serve ships traveling in direction d. In fact, the capacity of the large chamber suffices to serve all ships arriving at time t and traveling in direction d. Consequently, we can ignore the simultaneously arriving ships in what follows.

3. If $C_{\text{large}} < k_{t,d}$, replace all these ships by two ships traveling in direction d, arriving at time t. Further, pre-assign the first ship to the first chamber and the second ship to the second chamber. It follows that both chambers must be available at time t to serve ships traveling in direction d.

The resulting instance is called $\mathcal{I}_{\text{unit}}$. Finding a no-wait schedule for $\mathcal{I}_{\text{unit}}$ with pre-assigned ships thus yields a no-wait solution for the original instance $\mathcal{I}$ of NLS-2. We now describe a reduction to 2-SAT for the problem of finding a no-wait solution for instances with two unit-capacity chambers and pre-assigned ships.

Recall that a no-wait schedule exists if and only if each ship can be assigned to either the short or the long chamber such that, for each chamber, lockages do not overlap. For lock chamber $j \in \{1, 2\}$, it is easily seen that there is no overlap if and only if $|t_i - t_{i'}| \geq T_j$ for each pair of ships i and i' that are assigned to chamber j, and in addition $|t_i - t_{i'}| \geq 2T_j$ if ships i and i' travel in the same direction, as argued in Section 1. We create the following instance of 2-SAT: for each ship $i \in \{1, \ldots, n\}$, define a literal x_i. We will argue later that $x_i = \mathit{false}$ corresponds to assigning ship i to the short chamber, and $x_i = \mathit{true}$ corresponds to assigning ship i to the long chamber. Let the Boolean expression in conjunctive normal form consist of clauses described as follows. For each pair of ships i, i' $\in \{1, \ldots, n\}$:

1. if the ships travel in opposite direction and $|t_i - t_{i'}| < T_1$, add the clause $(x_i \vee x_{i'})$,
2. if the ships travel in the same direction and $|t_i - t_{i'}| < 2T_1$, add the clause $(x_i \vee x_{i'})$,
3. if the ships travel in opposite direction and $|t_i - t_{i'}| < T_2$, add the clause $(\neg x_i \vee \neg x_{i'})$,
4. if the ships travel in the same direction and $|t_i - t_{i'}| < 2T_2$, add the clause $(\neg x_i \vee \neg x_{i'})$.

Observe that, if $x_i = \mathit{false}$ where ship i is assigned to the short chamber and $x_i = \mathit{true}$ where it is assigned to the long chamber, $(\neg x_i \vee \neg x_{i'})$ suffices to prevent overlapping lockages for the long chamber whereas $(x_i \vee x_{i'})$ ensures that there is no overlap for the short chamber.

In addition, to enforce that all pre-assignments are respected, we add a clause containing only the literal x_i for all ships i that are pre-assigned to the long chamber, and a clause consisting of $\neg x_i$ for all ships i that are pre-assigned to the short chamber.

We claim that the existence of a truth assignment satisfying the Boolean formula is equivalent to the existence of a no-wait schedule. Indeed, given a truth assignment, we assign ship i to the short chamber if $x_i = \mathit{false}$ and to the long chamber if $x_i = \mathit{true}$. The definition of the clauses implies that no overlapping lockages exist for either chamber while each ship is assigned to a chamber, and hence we found a no-wait schedule. Also, the existence of a no-wait schedule immediately translates into a satisfying truth assignment. □

Note that the number of clauses in the 2-SAT instance described above, as well as the time needed to construct this instance, is quadratic in the number of ships. To find a no-wait solution for the NLS-2 problem, it follows that we can construct an instance of 2-SAT as described above, and use any algorithm for 2-SAT with a running time linear in the number of clauses. We can summarize this in the following theorem:

Theorem 3. *The bi-directional case for two arbitrary chambers, i.e. problem NLS-2, can be solved in $O(n^2)$ time.*

4 Identical chambers

We now focus on problem NLS-id, i.e. $C_j = C$ and $T_j = T$ for each $j \in \{1, \dots, m\}$. Again we assume that arrival times are given in sorted order (Section 1.5). In fact, we consider a more general optimization version of NLS-id, where we aim at finding the minimum number of chambers allowing a no-wait solution. We first show that this problem is a special case of coloring trapezoid graphs. In Section 4.2, we provide a description of a greedy procedure and argue that it always finds a no-wait schedule while using a minimum number of chambers. We then prove in Section 4.3 that this procedure can be implemented with an $O(n)$ running time for both the uni-directional as well as the bi-directional case.

4.1 Coloring trapezoid graphs

For the definition of trapezoid graphs we consider a pair of parallel lines, labeled *up* and *down*. A trapezoid between these lines is defined by two points per line. Let this construction of lines and trapezoids be called the trapezoid instance. A graph $G = (V, E)$ is called a trapezoid graph if there exists a trapezoid instance with $|V|$ trapezoids, each corresponding to a node in V, such that there is an edge in E connecting nodes u and v if and only if the trapezoids corresponding to u and v intersect. See Figures 11 and 12 for an example illustrating this definition. Felsner et al. (1997) discuss the coloring of trapezoid graphs and show that a proper coloring with minimum number of colors can be found in $O(n \log n)$ time.

The special case of this trapezoid graph coloring problem that we consider is the following: given a trapezoid instance where all trapezoids are identical isosceles triangles, find a proper coloring with a minimum number of colors. We denote this problem by TC. Notice that while the triangles are identical, their orientation may differ depending on which of the parallel lines contains a single point. In a trapezoid coloring instance, we say that a triangle is *up-oriented* if this triangle has a single point on the up line and two points on the down line; a triangle is *down-oriented* if it has a single point on the down line and two points on the up line.

We argue that we can reduce NLS-id to TC, and vice versa. As in Section 3, let $k_{t,d}$ denote the number of ships arriving at time t and traveling in direction d; we first argue that we can deal with simultaneous arrivals by transforming each instance $\mathcal{I}$ of NLS-id into an equivalent instance $\mathcal{I}_{\text{unit}}$ with unit capacity. In the remainder of Section 4, we can then restrict ourselves to instances where $C = 1$. Given $\mathcal{I}$, we construct $\mathcal{I}_{\text{unit}}$ as follows. For each $t \in \mathcal{T}$, replace the $k_{t,d}$ arrivals by $\lceil k_{t,d}/C \rceil$ arrivals at time t and traveling in direction d. Clearly, in $\mathcal{I}$, at least $\lceil k_{t,d}/C \rceil$ chambers are needed to serve these $k_{t,d}$ ships. It is also clear that any remaining capacity in the chosen lock chambers cannot be used to serve

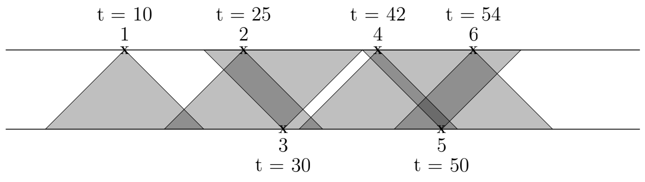

Figure 11: Example instance illustrating the definition of the corresponding trapezoids.

other ships without introducing waiting time. Consequently, a solution in $\mathcal{I}_{\text{unit}}$ corresponds directly to a solution in $\mathcal{I}$. Observe that constructing $\mathcal{I}_{\text{unit}}$ can be done in $O(n)$ time.

We thus turn our attention to the case with unit capacity. Given an instance of NLS-id where $C = 1$, we specify a set of identical isosceles triangles between parallel lines *up* and *down* as follows. For each downstream traveling ship i we construct a triangle with a point on the up line at t_i and points on the down line at $t_i - T$ and $t_i + T$; for each upstream traveling ship i we construct a triangle with a point on the down line at t_i and points on the up line at $t_i - T$ and $t_i + T$. From this construction, we can derive two fundamental properties:

1. The triangles corresponding to two ships i and i' traveling in the same direction intersect if and only if $|t_i - t_{i'}| < 2T$.

2. The triangles corresponding to two ships i and i' traveling in opposite directions intersect if and only if $|t_i - t_{i'}| < T$.

Note that in either case, ships i and i' cannot be served by the same chamber. Concluding, ships can be assigned to the same chamber if and only if the corresponding triangles do not intersect. Hence, a proper coloring of the corresponding graph represents a no-wait schedule where a color refers to a chamber. The chromatic number of the trapezoid graph, then, represents the minimum number of chambers allowing a no wait schedule. Each instance of NLS-id can thus be modeled as an instance of TC. Similarly, it is easily seen that each instance of TC can be modeled as an instance of NLS-id.

In order to illustrate this reduction we provide an example instance in Figure 11. We have downstream-traveling ships 1, 2, 4, and 6, arriving at times 10, 25, 42, and 54, and upstream-traveling ships 3 and 5, arriving at times 30 and 50. The lockage time is $T = 10$. Consequently, the pairs of ships that cannot both be served by a single chamber are (1,2), (2,3), (2,4), (4,5), (4,6), and (5,6). This is represented by intersections of triangles accurately. The graph corresponding to this instance is shown in Figure 12. It is immediately seen that at least three colors are required to color the graph since it contains a clique on nodes 4, 5, and 6. One feasible solution could consist of assigning ships 1 and 4 to the first chamber, ships 2 and 5 to the second chamber, and ships 3 and 6 to the third chamber.

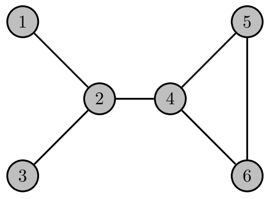

Figure 12: Trapezoid graph corresponding to the example instance of Figure 11.

Since the chromatic number of a trapezoid graph can be found in $O(n \log n)$ time, it immediately follows that an $O(n \log n)$ algorithm exists that solves NLS-id. In the remainder of this section, we improve this result to yield an $O(n)$ algorithm.

4.2 Correctness of a greedy procedure for NLS-id

We argued in Section 4.1 that we can reduce each instance of NLS-id to an equivalent instance with $C = 1$. We thus restrict ourselves to the setting with $C = 1$. We say that a chamber is available at time t for direction d if the last ship handled by this chamber (say ship i) traveled in direction d and arrived at time $t_i \leq t - 2T$, or traveled in the direction opposite to d and arrived at time $t_i \leq t - T$. In the former case, we say that the availability period of the chamber equals $t - t_i - 2T$; in the latter case, the availability period of the chamber equals $t - t_i - T$. The solution procedure is as follows. Initially, let the number of chambers be zero. Consider all arrival ships in order, let t_i and d_i be the arrival time and direction of ship i being considered. If no chambers are available at time t_i for direction d_i, add an additional chamber and assign ship i to this chamber; otherwise, assign ship i to the chamber with the largest availability period for direction d_i at time t_i.

We argue that this greedy procedure yields a solution with a minimum number of chambers. Whenever a chamber is added to serve ship i, there is no possible assignment of ships $1, \ldots, i - 1$ to chambers so that any of the chambers is available at time t_i. Indeed, since all chambers are identical, the choice of which preceding ship is assigned to which chamber is in fact irrelevant. Thus, after considering a ship i, a chamber is added if and only if the current number of chambers is not sufficient to serve ships $1, \ldots, i$ without waiting time. It follows that after considering all ships, we have a no-wait solution which uses a minimum number of lock chambers.

4.3 An $O(n)$ algorithm for NLS-id

To see that the procedure from Section 4.2 can be implemented to run in linear time, we first briefly discuss the uni-directional setting before extending the implementation to the bi-directional case. When all ships travel in the upstream direction, it is easily seen that there is an optimal solution where a chamber, after transferring a ship, immediately returns to the downstream side. A chamber that serves a ship is then always unavailable for $2T$ time units, starting from the arrival time of that ship. Solving this problem corresponds to a basic interval scheduling problem, for which Ford and Fulkerson (1962) describe a 'staircase rule' based on Dilworth's chain decomposition theorem. Gupta et al. (1979) provide a more efficient algorithm, which runs in $O(n)$ time when the intervals have equal length and are sorted by starting time. Applying this algorithm thus immediately yields a solution to NLS-uni-id.

While this approach is straightforward for NLS-uni-id, the time at which a chamber becomes available depends on the direction of travel in the more general NLS-id. Indeed, a chamber that finishes an upwards lockage is immediately available to serve a downstream traveling ship; the next upstream traveling ship, however, can only be served after an additional T time units needed to return to the downstream side. As a result, for a given time t and direction d, a chamber which started a lock movement at time t_1 may not be available while a different chamber which started a lock movement at time $t_2 > t_1$ is available. The main challenge of implementing our greedy rule is thus to efficiently keep track of the different chambers, and the moments in time at which they are available to serve ships depending on their direction.

To achieve this, we maintain the following lists throughout the solution procedure. Each of the entries that will be added to these lists consists of a pair (t, i), where t specifies the time at which the chamber that serves ship i becomes available for a given direction.

1. list A^{UU}: availability for upstream, ship i is upstream-traveling.
2. list A^{UD}: availability for upstream, ship i is downstream-traveling.
3. list A^{DU}: availability for downstream, ship i is upstream-traveling.
4. list A^{DD}: availability for downstream, ship i is downstream-traveling.

As outlined in the description in Section 4.2, we keep track of the required number of chambers m. Additionally, we follow up whether each chamber remains available as the algorithm runs. Throughout the algorithm, R_i are Boolean values indicating whether, after serving ship i, a chamber has been used to serve another ship ($1 \le i \le n$).

A pseudo-code for the algorithm is provided in Algorithm 2. In words: we consider each ship in order and first verify whether one of the existing chambers is available at the position where the ship arrives. Let i be the ship under consideration. If a chamber j is available, it contains an entry in two of the lists. Upon assigning a ship to j we update R_i so that the second entry corresponding to j becomes invalid. We update the times at which j becomes available for each direction. If no chamber is available, we update m and proceed as above.

To see that Algorithm 2 runs in linear time, note the following. Each ship is considered only once, in the given input order. New entries in the lists A^{UU}, A^{UD}, A^{DU}, and A^{DD} are always added to the end of the list. Furthermore, the time value of newly inserted entries is non-decreasing with i since the increment

```
input: arrival times $t_1 \leq t_2 \leq \ldots < t_n$, directions $d_1, d_2, \ldots, d_n$, lockage duration $T$
$A^{UU} \leftarrow \emptyset$, $A^{UD} \leftarrow \emptyset$, $A^{DU} \leftarrow \emptyset$, $A^{DD} \leftarrow \emptyset$
$R_i \leftarrow$ false, for all $i \in \{1, \ldots, n\}$
$m \leftarrow 0$
for $i = 1$ to $n$ do
    reUsed = false
    if $d_i == downstream$ then
        $(t^*, i^*) \leftarrow$ earliest entry in $A^{DU} \cup A^{DD}$
        while $t^* < t_i$ and $reUsed ==$ false do
            if $R_i ==$ false then
                reUsed = true
                $R_{i^*} \leftarrow$ true
            delete entry $(t^*, i^*)$ from the list in which it is contained
            $(t^*, i^*) \leftarrow$ earliest entry in $A^{DU} \cup A^{DD}$
    else
        $(t^*, i^*) \leftarrow$ earliest entry in $A^{UU} \cup A^{UD}$
        while $t^* < t_i$ and $reUsed ==$ false do
            if $R_i ==$ false then
                reUsed = true
                $R_{i^*} \leftarrow$ true
            delete entry $(t^*, i^*)$ from the list in which it is contained
            $(t^*, i^*) \leftarrow$ earliest entry in $A^{UU} \cup A^{UD}$
    if $reUsed ==$ false then
        $m \leftarrow m + 1$
    if $d_i == downstream$ then
        add entry $(t_i + T, i)$ to the back of list $A^{UD}$
        add entry $(t_i + 2T, i)$ to the back of list $A^{DD}$
    else
        add entry $(t_i + T, i)$ to the back of list $A^{DU}$
        add entry $(t_i + 2T, i)$ to the back of list $A^{UU}$
return $m$
```

Algorithm 2: Pseudo-code for identical chambers, i.e. NLS-id, with unit capacity.

when constructing the entry is the same for all entries within each of the lists; for example, all entries inserted in list A^{UU} have a time value of $t_i + 2T$ for some t_i, and all entries inserted in list A^{UD} have a time value of $t_i + T$ for some t_i. Each of the lists thus remains sorted by the time value of the contained entries at all times. Finding the earliest entry in two of the lists is then easily performed in constant time by comparing the first entry of each of the lists. Deleting the first entry as well as adding a new entry to the back of a list also require only constant time. Whenever an entry of one of the lists is iterated over, it is deleted. Since only $O(n)$ entries are added to lists throughout the procedure, iterating through the lists thus also takes $O(n)$ time in total. It follows that the entire procedure runs in linear time. We can summarize the discussion above as follows:

Theorem 4. *For the setting with identical chambers, a no-wait schedule can be found or shown not to exist, i.e. problem NLS-id can be solved, in $O(n)$ time.*

Corollary 1. *Finding the chromatic number of a trapezoid graph where the trapezoids are identical up- or down-oriented isosceles triangles can be done in $O(n)$ time.*

5 A dynamic programming algorithm for arbitrary m

In the following, we propose a dynamic programming (DP) approach that solves the general problem NLS, stated in Section 1. That is, we consider an arbitrary number of chambers m, with non-identical lockage times T_j and capacities C_j for $j = 1, \dots, m$. The proposed approach is similar to an $O(mn^{m+1})$ DP algorithm presented by Sung and Vlach (2005) for a parallel machine scheduling problem with deadlines and just-in-time jobs.

We assume ships to be numbered in non-decreasing order of arrival times. Ties are broken by letting ships arriving downstream have lower numbers than ships arriving upstream. Remaining ties are broken arbitrarily. Our DP approach assigns ships to chambers in increasing order of these numbers. We consider states $(i_1, \dots, i_m)$ where i_j represents the last ship $1 \leq i_j \leq n$ that has been assigned to chamber j $(1 \leq j \leq m)$. Furthermore, we restrict ourselves to states where a ship is assigned to a chamber only if all earlier ships have also been assigned. Thus, in a given state $(i_1, \dots, i_m)$, the first $\max_{j \in \{1,\dots,m\}} i_j$ ships have been assigned to chambers. We consider a transition from $(i_1, \dots, i_m)$ to $(i'_1, \dots, i'_m)$ if there is a $j^* \in \{1, \dots, m\}$ such that:

1. $i'_{j^*} > i_{j^*}$ and $i'_j = i_j$ for each $j \in \{1, \dots, m\}$ with $j \neq j^*$,
2. ships $(\max_j i_j) + 1, \dots, i'_{j^*}$ travel in the same direction and arrive at the same time,
3. i'_{j^*} arrives at least T_{j^*} time units later than i_{j^*} if both travel in opposite direction and i'_{j^*} arrives at least $2T_{j^*}$ time units later than i_{j^*} otherwise, and

4. $i'_j - (\max_j i_j) \leq C_{j^*}$.

This transition represents assigning ships $(\max_j i_j) + 1, \ldots, i'_{j^*}$ to chamber j^*. This is allowed only if chamber j^* is available after handling ship i_{j^*}. If more than one ship is assigned to chamber j^*, then these ships must arrive simultaneously, travel in the same direction, and the chamber's capacity must not be exceeded. We consider an initial state $(0, \ldots, 0)$ representing that no ships are assigned to any chambers yet. The question is whether we can reach a state $(i_1, \ldots, i_m)$ with $\max\{i_j \mid j = 1, \ldots, m\} = n$ by any sequence of transitions.

In this DP we use $O(n^m)$ states and $O(mn^{m+1})$ transitions. However, we can further restrict the set of transitions by always assigning the maximum number of ships (up to the chamber's capacity) which travel in the same direction and arrive at the same time as ship $i_{j^*} + 1$. Each state then has m transitions: one per chamber. This leaves us with $O(mn^m)$ transitions, which also constitutes the runtime complexity. Note that the complexity is polynomially bounded if the number of chambers is fixed. We can conclude with the following theorem:

Theorem 5. *The problem setting with a fixed number of arbitrary chambers, i.e. problem NLS-m, can be solved in $O(mn^m)$ time.*

6 Number of chambers m part of the input

We prove here that problem NLS is NP-complete.

Theorem 6. *Deciding whether a no-wait solution exists for an arbitrary number of chambers, even when all ships travel in the same direction and arrival times are distinct, i.e. problem NLS-uni-distinct, is strongly NP-complete.*

Proof. We prove the theorem by a reduction from numerical matching with target sums where all given integers are distinct. We will denote this problem as dNMTS. Hulett et al. (2008) showed that this special case of the classical NMTS problem is strongly NP-complete. In an instance of dNMTS we are given $3n$ pairwise distinct positive integers a_i, b_i, c_i $(1 \leq i \leq n)$ with $\sum_i^n c_i = \sum_i^n (a_i + b_i)$. The question is whether there exists a collection of n triples (i, j, k) such that, for each triple, $a_i + b_j = c_k$, and such that each integer in the input occurs in exactly one triple. We show that we may impose, without loss of generality, two additional constraints on the instances of dNMTS. Our assumptions are (i) $\max_i a_i - \min_i a_i < \min_i b_i$, and (ii) $\max_i c_i - \min_i c_i < \min_i b_i$. Assumption (i) is immediately seen to hold since the instances in the proof provided by Hulett et al. (2008) satisfy $\max_i a_i < \min_i b_i$. For assumption (ii), observe that we may transform any instance into an equivalent instance where $\max_i c_i - \min_i c_i < \min_i b_i$ by adding an arbitrary positive K to all b_i and c_i of the original instance. This does not change the answer to the decision question; also note that by adding any $K > 0$, all resulting values remain distinct, and assumption (i) remains valid. Since the left-hand side of the inequality in assumption (ii) does not change by this operation, we obtain the desired result by choosing any $K > \max_i c_i - \min_i c_i - \min_i b_i$.

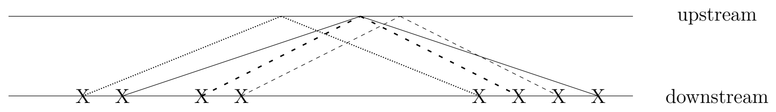

Figure 13: Graphical representation of the instance for the proof. Ship arrivals are marked with 'X', time passes from left to right. The tilted lines correspond to lock movements of the different chambers in a no-wait solution.

For any instance of dNMTS satisfying the two assumptions specified above, we now construct an instance of the lock scheduling problem with parallel chambers as follows. There are $2n$ ships arriving on the downstream side: n ships arrive at times $t_i = a_i$ and n ships arrive at times $t_{n+k} = c_k$, with $i, k \in \{1, \dots, n\}$. For convenience, we will refer to the first n arrivals as the set of ships A, and the last n arrivals as the set of ships C. There are $m = n$ chambers, each with a lockage time equal to $T_j = {}^{b_j}/_2$ for $j \in \{1, \dots, m\}$. The question remains whether a no-wait solution exists for this instance. Figure 13 shows a graphical representation. Next, we show that the given instance of dNMTS is a 'yes' instance if and only if there exists a no-wait solution to the constructed lock scheduling instance.

If there is a solution to the instance of dNMTS, each triple (a_i, b_j, c_k) corresponds to a combination of a chamber with one ship from the set A, and one ship from the set C. If the ship in A enters the chamber with lockage time ${}^{b_j}/_2$ at time a_i, the ship in C can enter the same chamber at time $c_k = a_i + 2\,({}^{b_j}/_2)$. Neither ship incurs any waiting time. Since each ship in A and C corresponds to exactly one such triple, there exists a no-wait solution.

On the other hand, we argue that if a no-wait solution exists, there must also exist a corresponding set of triples that satisfies the requirement of the dNMTS problem. We first observe that in all no-wait solutions, each chamber handles exactly one ship from A, and one ship from C. Indeed, since all arrival times are distinct, a chamber cannot simultaneously transfer more than one ship without introducing waiting time. Further, it follows from $\max_i a_i - \min_i a_i < \min_i b_i$ that no chamber can serve two ships from A so that no ships incur waiting time. Similarly, it holds that no chamber can serve two ships from C without introducing waiting time. Thus, in a given no-wait solution, each chamber transfers exactly one ship from A and one ship from C. For each chamber j, let $i(j)$ be the index of the ship from A and $k(j)$ the index of the ship in C that is handled by chamber j. We thus obtain n triples $(a_{i(j)}, b_j, c_{k(j)})$. Since the solution is no-wait, we have $a_{i(j)} + b_j \le c_{k(j)}$ for all j. Finally, because $\sum_i^n (a_i + b_i) = \sum_i^n c_i$, it is clear that if $a_{i(j)} + b_j < c_{k(j)}$ for any j, there must exist a j' such that $a_{i(j')} + b_{j'} > c_{k(j')}$, which would mean that at least one ship incurs waiting time. It follows that $a_{i(j)} + b_j = c_{k(j)}$ for all $j \in \{1, \dots, n\}$, and there thus exists a set of triples that identifies the given instance of NMTS as a 'yes' instance. This concludes the proof. □

As mentioned in the introduction, when viewing a chamber as a machine and an arrival as a job represented by m intervals (one for each machine, each starting at the same moment t_i), the above reduction shows that the problem considered by Böhmová et al. (2013) (called Interval Selection with cores) remains NP-complete even when all intervals that correspond to the same machine have the same length.

Acknowledgment

This research has been supported by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office.

References

Aspvall, B., Plass, M. F., and Tarjan, R. E. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121 – 123.

Böhmová, K., Disser, Y., Mihalák, M., and Widmayer, P. (2013). Interval selection with machine-dependent intervals. In *WADS'13*, pages 170–181.

Disser, Y., Klimm, M., and Lübbecke, E. (2015). Scheduling bidirectional traffic on a path. In Halldórsson, M. M., Iwama, K., Kobayashi, N., and Speckmann, B., editors, *Automata, Languages, and Programming*, volume 9134 of *Lecture Notes in Computer Science*, pages 406–418. Springer Berlin Heidelberg.

Even, S., Itai, A., and Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703.

Felsner, S., Müller, R., and Wernisch, L. (1997). Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32.

Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks.* Princeton University Press.

Gupta, U., Lee, D., and Leung, J.-T. (1979). An optimal solution for the channel-assignment problem. *Computers, IEEE Transactions on*, C-28(11):807–810.

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

Hulett, H., Will, T. G., and Woeginger, G. J. (2008). Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36:594–596.

Kolen, A. W., Lenstra, J. K., Papadimitriou, C. H., and Spieksma, F. C. (2007). Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543.

Krumke, S. O., Thielen, C., and Westphal, S. (2011). Interval scheduling on related machines. *Comput. Oper. Res.*, 38(12):1836–1844.

Passchyn, W., Briskorn, D., and Spieksma, F. C. R. (2016). Mathematical programming models for lock scheduling with an emission objective. *European Journal of Operational Research*, 248(3):802 – 814.

Passchyn, W., Coene, S., Briskorn, D., Hurink, J., Spieksma, F. C. R., and Vanden Berghe, G. (2015). The lockmaster's problem. *European Journal of Operational Research*.

Prandtstetter, M., Ritzinger, U., Schmidt, P., and Ruthmair, M. (2015). A variable neighborhood search approach for the interdependent lock scheduling problem. In Ochoa, G. and Chicano, F., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 9026 of *Lecture Notes in Computer Science*, pages 36–47. Springer International Publishing.

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

Sung, S. C. and Vlach, M. (2005). Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *J. of Scheduling*, 8(5):453–460.

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

Verstichel, J. (2013). *The Lock Scheduling Problem*. PhD thesis, KU Leuven.

Verstichel, J., De Causmaecker, P., Spieksma, F., and Vanden Berghe, G. (2014). The generalized lock scheduling problem: An exact approach. *Transportation Research E, Logistics and Transportation Review*, 65:16–34.

Waterwegen en Zeekanaal NV and nv De Scheepvaart (2014). *Masterplan voor binnenvaart op de Vlaamse waterwegen - Horizon 2020 [Master plan for inland shipping on the Flemish waterways - Horizon 2020]*. Willebroek: Waterwegen en Zeekanaal NV, Hasselt: nv De Scheepvaart. (in Dutch).

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