Ship Traffic Optimization for the Kiel Canal

Elisabeth Lübbecke∗ Marco E. Lübbecke† Rolf H. Möhring∗

November 28, 2014

Abstract

We introduce, discuss, and solve a hard practical optimization problem which we call the ship traffic control problem (STCP). Since we plan bi-directional traffic, STCP relates to, and in fact generalizes train timetabling on single-track networks. The objective of finding quickest routes motivates the integration of recent algorithmic ideas from dynamic collision-free routing of automated guided vehicles. We offer a unified view of routing and scheduling which blends simultaneous (global) and sequential (local) solution approaches to allot scarce network resources to a fleet of vehicles in a collision-free manner. This leads us to construct a fast online heuristic.

The STCP originates from the Kiel Canal which is the basis for the trade between the countries of the Baltic area and the rest of the world. As traffic is projected to significantly increase, the canal is planned to be enlarged in a billion Euro project. Our work forms the mathematical and algorithmic basis for a tool to evaluate the different enlargement options. In view of computational experiments on real traffic data expert planners approved that our combinatorial algorithm is well-suited for this decision support. With the help of instance-dependent lower bounds we assess the quality of our solutions which significantly improves upon manual plans.

We are confident that our ideas can be extended to other application areas like train timetabling and collision-free routing, also in more general networks.

1 Introduction

With more passages than the Panama and Suez Canals together, the Kiel Canal is the world’s busiest artificial waterway. It is located in the north of Germany and links the North and Baltic Seas. An average of 250 nautical miles (460km) is saved by using the canal instead of the way around Skaw. The Kiel Canal, as the more ecological and safer route, became the basis for the trade between the countries of the Baltic area with the rest of the world [14].

The canal is operated bi-directionally. Since offshore vessels are not primarily designed for inland navigation, the passing of two ships with large dimensions is not possible at arbitrary positions. This is called a conflict. To resolve these, there are dedicated locations within the canal, called sidings or turnouts, which are wider to allow for passing and waiting. Decisions must be made about who is waiting for whom, where, and for how long. This traffic control

∗Institut für Mathematik, Technische Universität Berlin, Str. d. 17. Juni 136, D-10623 Berlin, Germany. Email: eluebbecke@math.tu-berlin.de (corresponding author), Rolf.Moehring@tu-berlin.de; supported by the DFG Research Center Matheon Mathematics for Key Technologies in Berlin.
†Operations Research, RWTH Aachen University, Kackerstr. 7, D-52072 Aachen, Germany. Email: marco.luebbecke@rwth-aachen.de
is done by a higher authority, the Waterways and Shipping Board with a team of nautically experienced expert navigators. They try to distribute waiting times in sidings fairly among all ships. The objective is to minimize the total transit times of all ships.

Since the end of the 1990s, the Kiel Canal has seen a tremendous growth of traffic demand. Not only the absolute number of vessels is growing, but also the total gross tonnage per ship [14]. As this development is expected to continue, the Kiel Canal may become inoperable when no countermeasures are taken: it is the growing share of large vessels which entail more of the above mentioned conflicts and challenge the navigators. As a remedy, an enlargement of the Kiel Canal is planned, with construction options such as extending or creating sidings or to allow more flexible passing of ships by deepening and/or widening crucial parts of the canal. In order to assess the cost and benefit of these options their combined effects under predicted ship traffic needed to be reliably estimated. In this paper we develop the mathematical and algorithmic foundation that leads to an optimization tool which emulates the current ship traffic control. This tool was used to evaluate the various construction options with the aim of selecting a most adequate combination.

Algorithmic Overview. The ship traffic control problem (STCP) contains a combinatorial and a geometric component. We first discuss the geometry (Section 2.2) as it completely describes the feasibility of itineraries in space and time. In order to capture the combinatorics we consider a relaxation which ignores capacities in sidings and focuses on resolving precedence conflicts by scheduling decisions (Section 3.1). While itineraries (geometry) imply scheduling decisions (combinatorics), the converse is not true in general. However, for the relaxation, we show that itineraries can be constructed from a feasible schedule in polynomial time by a dynamic routing algorithm that treats only a single ship at a time (Theorem 1). This quickest path style algorithm extends to the general situation to insert a single route into an existing plan, respecting all constraints imposed by the geometry, when this is possible (Section 4). Yet, this algorithm alone is inappropriate for iteratively constructing solutions by routing ships one by one: we show that any such sequential routing procedure may yield a total travel time arbitrarily far from optimum (Lemma 2). The reason lies in the inherent interdependence of the problem’s geometry and combinatorics. We thus devise a local search on the scheduling decisions (Section 5.2) which uses a routing algorithm to incorporate the geometric part. We finally reflect the problem’s online character (only a limited amount of future routing requests is known) by embedding the algorithm into a rolling horizon planning (Section 5.3).
Our Contributions. Although our application context is quite particular, several methodological aspects are of broader interest.

1. We merge algorithmic ideas from two related applications that deal with collision-avoidance: On the one hand, train timetabling on a single-track railway network has a common interpretation as a job-shop scheduling problem. This relation also influenced our modeling and solution approaches. On the other hand, collision-free routing of automated guided vehicles has spawned the development of elaborate quickest path algorithms. Both aspects, the scheduling and the dynamic routing, are interdependently present in our traffic control problem. We show how a dynamic routing algorithm for a single vehicle can be enhanced with scheduling decisions such as to produce a collision-free routing for a whole fleet of vehicles.

2. Time-space, or time-expanded networks are a standard way to model routing problems over time. In practice, with a fine time discretization, these networks quickly grow beyond any manageable size. Our application additionally called for a fine space discretization, rendering such a modeling approach infeasible. We have shifted the information complexity from the network representation to the routing algorithm, resulting in an extremely simple network, but a more involved algorithm: We propose a Dijkstra-like shortest path algorithm that implicitly handles time and space discretization with an arbitrary precision. It is fair to say that, implicitly, our algorithm chooses, out of a continuum of possible discretizations, exactly the right one.

3. Collision-free routing is a research topic of wide practical interest; besides rail-bound transportation, applications like scheduling industrial robots [21, 23] come to mind. In order to avoid collisions it is customary to block (parts of) trajectories for others for the whole time interval it in use by one robot, even if this is not required in practice. We only forbid entry times into these trajectories which enables our implicit space discretization. This allows having several robots on the same trajectory simultaneously and resolves an algorithmic problem which hitherto wasted non-negligible optimization potential.

4. The geometric and combinatorial problem components naturally suggest a decomposition based approach. While complex geometric feasibility constraints hinder the definition of a neighbor in a local search, combinatorial neighbors can be easily computed. We use our dynamic routing algorithm to turn them into improved geometries.

5. Last but not least, we solve a very complex practical problem to the fullest satisfaction of the problem owners. Routing over time, scheduling, and (rectangle) packing aspects are to be considered simultaneously, and in an online manner. We developed a decision support tool to evaluate the control of current and future traffic scenarios on the Kiel Canal, respecting numerous constraints at an enormous level of detail. The quality of our planning was verified by experts at different levels. Thus, our tool was used in a study to decide about the enlargement options for the Kiel Canal.

2 The Ship Traffic Control Problem (STCP)

2.1 The Planning Context

Any two ships, no matter what size, can pass each other in sidings. Outside sidings, on transit segments, legal regulations and nautical parameters precisely define the circumstances under which ships are allowed to meet. Ships are categorized into traffic groups 1–6, mainly depending on their dimensions and charge; transit segments are classified into passage numbers 6, 7,
or 8, reflecting physical dimensions. In simplified terms, two ships may pass each other on a transit segment only if the sum of their traffic group numbers does not exceed the respective passage number, see Figure 2. Also, overtaking a moving ship is considered a dangerous maneuver and thus is not an option.

Figure 2: A transit segment of passage number 6. The ship of traffic group 4 has to wait in a siding until the ship of traffic group 3 has released the segment (reason: $3 + 4 > 6$). The ship of traffic group 2 can pass (reason: $3 + 2 \leq 6$).

To protect the river bed, velocities of most of the largest ships are limited to 12km/h, that of all others to 15km/h. We assume constant full speed for each ship, as planners do. Because of non-uniform velocities we also need to obey location dependent safety distances.

In each of the 12 sidings situated at the Kiel Canal, in each direction, there is a passage track and an additional waiting track, see Figure 2. A ship travels on the passage track to its waiting position and instantaneously changes over to the waiting track where it stays for the full duration of waiting, then moves back to the passage track. In addition to the ship’s length safety distances to other ships must be kept, both, when waiting and moving. Overtaking a waiting ship is possible. Under very restricted conditions it is allowed to wait on the waiting track of the opposite direction. Obviously, the waiting capacity of a siding is limited, and the choice of waiting positions influences how well this capacity is used. As a rule of thumb, for each ship, the expert planners try not to exceed a waiting time of three hours during the whole journey and one and a half hours individually in each siding. For ships of traffic group 6 this reduces to two hours in total and one hour per siding.

A journey through the canal typically takes seven to nine hours. It may be preempted on many occasions, into side arms or berths. Hence, ships can appear or disappear for planning literally on arbitrary positions within the canal.

Current traffic control is aided by a distance-time diagram (also: time-space or string-line diagram, known from traffic timetabling [3]). Figure 3 illustrates an example, a screen-shot of the diagram actually in use is given in Figure 4.

We finally stress the online character of ship traffic control. There is a limited look-ahead, as ships register quite late for a journey through the canal only, about two hours before arrival. Because of a scheduled locking process at the canal’s boundaries, arrival times of ships at the first siding are known and preliminary itineraries can be planned. However, these need to be updated upon arrival of new ships.

### 2.2 A Precise Geometric Model

A canal is represented as an interval $C \subseteq \mathbb{R}$, partitioned into a set $E$ of $m$ intervals, called segments, i.e., $\bigcup_{E \in E} E = C$. Elements of a distinguished subset $T \subseteq E$ are called sidings; we refer to all other segments as transit segments. A set of ships $S = \{1, \ldots, n\}$ induces a set of requests $R = \{(s_i, t_i, v_i, r_i, h_i) \mid i \in S\}$ with start and target $s_i \neq t_i \in C$ at arbitrary positions in the canal, a velocity $v_i$, a release time $r_i$, and a parking distance $h_i$ (see below).
The travel direction of a ship (its *heading*), or simply the ship $i \in S$ itself is called *upstream* if $t_i > s_i$ and *downstream* otherwise. Two ships heading in the same direction are called *aligned*, otherwise *opposed*. The velocity of a ship $i \in S$ and the length of a segment $E \in \mathcal{E}$ define the transit time $\tau_{iE}$ of ship $i$ along segment $E$. For each pair of ships $(i,j) \in S \times S$ and each segment $E \in \mathcal{E}$ we are given a *vertical distance* $v_{ijE} \geq 0$. It ensures a sufficient headway between two (aligned or opposed) ships, as defined by the sizes of the ships and the required safety distance (translated to the corresponding transit time via the velocities). Since the safety distance between two aligned ships can depend on the size of the ship following behind, $v_{ijE} \neq v_{jiE}$ is possible. The *horizontal parking distance* $h_i \geq 0$ of each ship $i \in S$ defines the required space when the ship is waiting in a siding. Figure 5 illustrates a small problem instance.
For each ship, decisions must be taken concerning space within the canal and time, simultaneously. We call this finding dynamic routes. When speaking of a ship’s position we actually refer to the ship’s center, without further notice. A solution is given by a set of simple polygonal chains in $C \times \mathbb{R}$, or polylines, one for each request, $P = \{P_i = (p_{i,1}, \tau_{i,1}), \ldots, (p_{i,k_i}, \tau_{i,k_i}) | i \in S\}$, such that the following properties hold. For each ship $i \in S$, polyline $P_i$ defines a dynamic route from the ship’s start position and release time to its target position, i.e., $p_{i,1} = s_i$, $\tau_{i,1} = r_i$ and $p_{i,k_i} = t_i$. The slopes are defined by the ship’s velocity, the slope’s sign reflects the ship’s heading, and waiting (zero velocity, infinite slope) is allowed only in sidings, i.e., either $\frac{p_{i,k} - p_{i,k-1}}{\tau_{i,k} - \tau_{i,k-1}} = v_i$ with $(p_{i,k} - p_{i,k-1})(t_i - s_i) > 0$ or $p_{i,k} = p_{i,k-1} \in \bigcup_{T \in T} T$ with $\tau_{i,k} > \tau_{i,k-1}$ for each breakpoint $k = 2, \ldots, k_i$. A ship is allowed to wait in each siding at most once, i.e., a polyline’s number of breakpoints per segment is bounded by 2. The waiting time of ship $i \in S$ in a solution $P$ is defined as the difference of $i$’s actual passage time in $P$ and $i$’s minimum possible passage time, i.e., $w_i := (\tau_{i,k_i} - \tau_{i,1}) - |t_i - s_i|/v_i$. A feasible (or synonymously collision-free) solution $P$ satisfies the following two properties, cf. Figure 6.

**Passing rules:** Line segments corresponding to two moving ships $i, j \in S$ (i.e., two segments of finite slope) are not allowed to be too close to each other (or even to cross) if $i$ and $j$ are aligned (in particular, no overtaking) or if $i$ and $j$ are opposed and their passing is not allowed at the respective position. In both cases this will be ensured by the minimal vertical distance $v_{ij,E}$ between the line segments.

**Parking distance:** The required space of a waiting ship defined by the horizontal parking distance $h_i$ must lie completely within the siding. Furthermore, this space must not be occupied by a second ship which is waiting concurrently.
In general, it is strongly NP-hard to decide if a feasible solution to the knapsack relaxation is identical the weaker knapsack condition suffices to ensure a corresponding rectangle packing. Our model of parking in sidings is stronger than simply requiring that the length of a ship does not exceed by parking ships, at no point in time. We refer to this weaker model as a feasible solution also as geometry.

In order to express feasibility mathematically, we need the following definitions. The map \( p_i : [\tau_{i,1}, \tau_{i,k_i}] \to C \) assigns each point in time \( \tau \in [\tau_{i,1}, \tau_{i,k_i}] \) to the corresponding position of ship \( i \in S \) within the canal at time \( \tau \) as defined by the polyline \( P_i \). For each ship position \( p \in C \), the inverse \( p_i^{-1}(p) =: \tau_i(p) \) gives a time interval, probably empty or a single point. The open set \( H_i(p) := \{[p - h_i, p + h_i] \times \tau_i(p)\}^o \), which is the interior of the rectangle occupied for potential waiting at position \( p \), is non-empty only if waiting actually takes place at point \( p \). Let \( \tau_i(p) := \max \tau_i(p) \) denote the latest point in time at which ship \( i \) stays at position \( p \), and similarly \( \tau_i(p) := \min \tau_i(p) \) denotes the earliest point in time, respectively. Abusing notation, we will reference the interval \([\min\{a,b\}, \max\{b,a\}] \subset C \) by \([a,b]\) when both, \( a \leq b \) or \( b \leq a \) is possible, depending on the currently considered heading. Mathematically, a solution is feasible or collision-free if the following holds:

**Passing rules:** For all \( i \neq j \in S, E \in \mathcal{E} \) and \( p \in E \cap [s_i, t_i] \cap [s_j, t_j] \) with \( \tau_i(p) \leq \tau_j(p) \):

\[
\tau_j(p) - \tau_i(p) \geq v_{ij} \cdot E .
\]

**Parking distance:** For all aligned ships \( i \neq j \in S, T \in \mathcal{T} \) and \( p_1, p_2 \in T \):

\[
H_i(p_1) \subset T \times \mathbb{R} \quad \text{and} \quad H_i(p_1) \cap H_j(p_2) = \emptyset .
\]

Remark. Our model of parking in sidings is stronger than simply requiring that the length of a siding is not exceeded by parking ships, at no point in time. We refer to this weaker model as knapsack relaxation (one may assume that all ships wait at the end of a siding). Our stronger model is much related to orthogonal rectangle packing on a strip [1]. If all ship dimensions are identical the weaker knapsack condition suffices to ensure a corresponding rectangle packing. In general, it is strongly NP-hard to decide if a feasible solution to the knapsack relaxation implies a corresponding feasible rectangle packing; see [2, 9, 13] for discussions and proofs.
In what follows, when speaking about time, it helps to define a common reference point. To this end, consider a ship \( i \in S \) at position \( p_1 \) at time \( \tau_1 \) and determine for some position \( p_2 \) the corresponding point in time \( \tau_2 \) on the straight line through \((p_1, \tau_1)\) with slope defined by the heading and velocity of \( i \) (e.g., the dashed lines in Figures 5 and 6). Formally, given a time value \( \tau_1 \) w.r.t. position \( p_1 \) for an upstream (resp. downstream) ship \( i \) we define \( \tau_2 := \tau_1 + (p_2 - p_1)/v_i \) (resp. \( \tau_1 - (p_2 - p_1)/v_i \)) to be the corresponding time translated to position \( p_2 \).

2.3 Related Work

The STCP has many similarities to train timetabling where arrival and departure times of train lines at stations or junctions have to be determined to avoid collisions. A recent overview on the large body of literature is [18]. In general, the common solution approaches use multi-commodity flow formulations in time-expanded networks with additional packing constraints. Prior works for special network structures representing long stretches of single and parallel track sections developed models asking for schedules of trains on each track section. This scheduling interpretation for single track networks was first used by [25] and spawned certain exact and heuristic methods, mostly based on LP techniques. Apart from the presence of arbitrary start and target positions on segments in the STCP the most significant difference to train timetabling occurs on the transit segments since the passing of two ships is forbidden for some pairs of ships but not for all. From the scheduling point of view the STCP can be considered as a generalization of scheduling with family setup times [20] and flow shop or restricted job shop scheduling [16]. In fact, the bi-directional character of the problem also yields new interesting theoretical research issues, see [8] for detailed discussions and complexity results.

The train timetable generation often works on aggregated networks and postpones the detailed routing of trains within junctions or stations to a separate planning step. To integrate the STCP’s modeling of waiting within sidings we refer to another related application which is the planning of automated guided vehicles (AGVs) described in [12]. There, routing requests for single vehicles arrive online over time and quickest dynamic routes without collisions to any of the previously planned vehicles must be computed. A similar collision-free dynamic routing approach can help to route the ships sequentially, one after another. Such a sequential application of dynamic routing optimizes only the arrival time of the currently considered ship and is not able to take care of the global objective. To deal with this drawback, we combine the routing and the scheduling view. It is important to note that most approaches in the literature use (time and/or space) discretization already at the modeling stage and thus take a loss in accuracy. This is systematically avoided by our geometric description.

3 Scheduling on Transit Segments

Having discussed the problem’s geometry in the previous section, we now focus on its combinatorics. We first consider a relaxation which ignores all constraints in sidings, in particular capacities, and thus concentrates on the combinatorial decisions to be taken on transit segments. Throughout this section, we assume that all parking distances and all vertical distances for sidings are zero, i.e., \( v_{ijT} = 0 \) for all \( i, j \in S \) and \( T \in T \), and \( h_i = 0 \) for all \( i \in S \). Already this combinatorial relaxation is NP-hard by a reduction from single-machine scheduling [8].
3.1 Understanding the Combinatorics

Ignoring all constraints in sidings, we can derive a solution just from the time a ship enters each segment, as each waiting position becomes feasible. A mixed integer program (MIP) will help us describe the relaxation, see Figure 7. Let $E_i^+$ denote the segment a ship must pass after segment $E \in \mathcal{E}$ when heading in the same direction as ship $i \in S$. Visit time variables $d_{iE}$ specify when ship $i \in S$ enters segment $E \in \mathcal{E}$. On transit segments, visit time variables are linked by the transit time $\tau_{iE}$ in Equation (2). On siding segments $T \in \mathcal{T}$, waiting is allowed which is reflected by waiting time variables $w_{iT} \geq 0$ for each ship $i \in S$. Equation (3) links visit and waiting times in sidings. Lower bounds (6) on visit times enforce release times.

| minimize $\sum_{i\in S, T\in \mathcal{T}} w_{iT}$ | (1) |
| s.t. $d_{iE} + \tau_{iE} = d_{iE}^+$ $\forall i \in S, E \in \mathcal{E} \setminus \mathcal{T}$ | (2) |
| $d_{iT} + \tau_{iT} + w_{iT} = d_{iT}^+$ $\forall i \in S, T \in \mathcal{T}$ | (3) |
| $z_{ijE} = 1 \implies d_{iE} + \Delta(i, j, E) \leq d_{jE} \quad \forall E \in \mathcal{E} \setminus \mathcal{T}, (i, j) \in C_E$ | (4) |
| $z_{ijE} = 0 \implies d_{jE} + \Delta(i, j, E) \leq d_{iE} \quad \forall E \in \mathcal{E} \setminus \mathcal{T}, (i, j) \in C_E$ | (5) |
| $d_{iE} \leq d_{iE}^+$ $\forall i \in S, E \in \mathcal{E}$ | (6) |
| $0 \leq w_{iT}$ $\forall i \in S, T \in \mathcal{T}$ | (7) |
| $z_{ijE} \in \{0, 1\}$ $\forall E \in \mathcal{E} \setminus \mathcal{T}, (i, j) \in C_E$ | (8) |

Figure 7: Mixed integer program for a combinatorial relaxation where all feasibility conditions concerning passing and waiting in sidings are relaxed. Constraints (4) and (5) can be linearized using a “big M” formulation.

Complications arise when ships $i$ and $j$ are not allowed to pass (when opposed) or overtake (when aligned) each other on segment $E$, i.e., they have a conflict on $E$ (this is equivalent to $v_{ijE} > 0$). We say that ship $i$ precedes ship $j$ if ship $i$ passes each position of segment $E$ before ship $j$, i.e., $\pi_i(p) < \pi_j(p)$ for all $p \in E \cap [s_i, t_i] \cap [s_j, t_j]$. For a conflicting pair, it must be decided, which ship precedes the other. Figure 8 shows that precedence can be quite involved. Define $\Delta(i, j, E)$ (or simply $\Delta(i, j)$ if the segment is obvious) to be the smallest value such that the condition

$$d_{iE} + \Delta(i, j, E) \leq d_{jE}$$

ensures the required vertical distance $v_{ijE}$ when ship $i$ precedes ship $j$ on segment $E$. The $\Delta$ need not be symmetric, see Figures 8(b) and 8(c). Figure 8(c) also demonstrates that the visit time $d_{iE}$ of the preceding ship $i$ can even be later than the visit time $d_{jE}$ of the succeeding ship $j$. Hence, the $\Delta$ values can be negative. Also in this case, Condition (9) defines a lower bound for $d_{jE}$ depending on the value of $d_{iE}$.

Denote by $C_E \subseteq S \times S$ the set of all conflicting pairs of ships on $E$. Depending on the decision which ship precedes on a segment, either Condition (9) must hold as stated, or with the roles of ships $i$ and $j$ reversed. This precedence or scheduling decision is represented by a binary variable $z_{i,j,E}$ and Implications (4) and (5) precisely reflect the above dichotomy. An assignment of the $z$ variables defines for each transit segment $E \in \mathcal{E} \setminus \mathcal{T}$ a partial order $\rho(E) \subset S \times S$ on the set of ships where $(i, j) \in \rho(E)$ if and only if $(i, j) \in C_E$ and ship $i$ precedes ship $j$. 


on segment $E$, i.e., $z_{ijE} = 1$. We refer to the collection (not the union) $\{\rho(E) \mid E \in \mathcal{E} \setminus \mathcal{T}\} := \rho$ of these partial orders as the \textit{combinatorial structure} of a solution or its \textit{combinatorics} for short. The combinatorics is the hard part of this relaxation: Once the precedences are decided on all segments, i.e., $z_{ijE}$ variables are fixed in MIP (1)–(8), an optimal assignment of the visit time variables (a geometry) can easily be obtained by solving the resulting linear program, or proven to be infeasible if no assignment exists. In the former case, we refer to each corresponding geometry as \textit{realization} of the given combinatorics and call the combinatorics itself \textit{realizable}.

\textbf{Remark.} The LP relaxation of MIP (1)–(8) is very weak. Since $z_{ijE} = 0.5$ for all $i, j, E$ is a feasible fractional solution, one obtains the trivial lower bound of zero. The model is still well suited to understand the combinatorial structure of the problem and good enough for first experimental comparisons. We therefore do not discuss models with stronger relaxations here.

\textbf{Remark.} We define visit time variables $d_{iE}$ for each ship $i \in S$ and for each segment $E \in \mathcal{E}$ even if the start point of $E$ is not included in the request interval $[s_i, t_i]$. This is done via translation of the time values between the interval boundaries and the start point of $E$ (w.r.t. $i$’s velocity and heading). That is, we think of each ship as traversing the whole canal, even though conditions concerning conflicts are active precisely on the ship’s request interval only.

\textbf{Remark.} The fact that ships can start and end virtually everywhere in the canal entails a variety of configurations of precedence, see again Figure 8. Our introduction of the $\Delta$ values enables us to use just one unified statement, namely Condition (9), to ensure precedence between conflicting ships. Moreover, this gives us a way to consider only a segment’s boundary to ensure precedence on the whole segment.

### 3.2 Realizing a Combinatorial Structure via Iterated Routing

We have just seen that optimal dynamic routes can be constructed by linear programming, once the scheduling decisions are fixed. However, as we show next, this \textit{global} view, i.e., \textit{simultaneously} deciding about all visit times is not necessary, but rather \textit{local} decisions for a single ship at a time suffice to create optimal collision-free dynamic routes for our combinatorial relaxation.

Assume we are already given dynamic routes $\mathcal{P}^*$ for a subset $S^* \subseteq S$ of ships, and we would like to construct a dynamic route for another ship $i \notin S^*$, respecting $\mathcal{P}^*$. This can be done in a \textit{greedy} manner. Define the visit times of $i$ consecutively for each segment as early as possible after its arrival without violating vertical distances to dynamic routes in $\mathcal{P}^*$. This yields a collision-free dynamic route with minimum waiting time for ship $i$ w.r.t. $\mathcal{P}^*$ and takes polynomial time. When a conflict occurs on a transit segment $E$, traversing $E$ as early.
as possible implies waiting at the end of a siding, directly in front of $E$. This is feasible as constraints regarding parking and vertical distances are relaxed in sidings.

We focus on solutions with a certain structure motivated by this greedy routing. Therefore, we call a dynamic route $P_i$ of a considered solution $\mathcal{P}$ to be *earliest arrival* w.r.t. the other dynamic routes of $\mathcal{P} \setminus \{P_i\}$, if the arrival time $\tau_{i}(p)$ of the dynamic route $P_i$ is at each position $p \in [s_i, t_i]$ as early as possible with respect to $\mathcal{P} \setminus \{P_i\}$ preserving collision-freeness and the solution’s combinatorics. Hence, the corresponding ship waits at no position longer than necessary, and all waiting of this ship takes place as late as possible. Note that the earliest arrival route of a ship with respect to the others and their combinatorics is unique. A solution $\mathcal{P}$ is called *earliest arrival* if each dynamic route of $\mathcal{P}$ is earliest arrival w.r.t. the other dynamic routes of $\mathcal{P}$. A solution can be turned into an earliest arrival solution by iteratively turning dynamic routes into earliest arrival ones in an arbitrary order (that may contain cycles) without increasing the waiting time of any ship. Observe that greedy routing produces an earliest arrival route for a single ship with respect to the already planned ones. Furthermore, greedy routing can be extended to respect additionally given predecessors of the current ship on each segment by defining the visit times of each segment as early as possible without violating them. We use this fact to produce earliest arrival solutions only given the combinatorics.

**Theorem 1.** For our combinatorial relaxation, there is a polynomial time algorithm using greedy routing that, given an instance $(C, S, R)$ and a combinatorial structure $\rho$ on $S$, either creates a feasible earliest arrival solution $\mathcal{P}$ realizing $\rho$ or proves that no realization exists.

We constructively prove the theorem by describing Algorithm 1. In Section 5.2 we generalize it to respect siding constraints. Define a *partial dynamic route* of ship $i \in S$ to be a dynamic route of $i$ with the relaxation that the first and the last $x$-coordinates can be any points in $[s_i, t_i]$, not necessarily on the boundary. A partial dynamic route of ship $i \in S$ is called a *prefix route* if the first $x$-coordinate equals $s_i$ and a *suffix route* if the last $x$-coordinate equals $t_i$. Consider some $(j, i) \in \rho(E)$ for a transit segment $E$, i.e., ship $j$ has to precede ship $i$ on $E$. To ensure this precedence the visit time of $j$ on $E$ must be known before deciding about the visit time of $i$ on $E$ via greedy routing. This can be achieved by producing partial dynamic routes for each ship in distinct steps, that fit together, instead of one complete dynamic route in one step. This is done in Line 4 of Algorithm 1 which works as follows:

1. Let $p_i$ be the last $x$- and $\tau_i$ be the last $y$-coordinate of the current partial dynamic route $P_i$ of $i$.
2. Find a collision-free suffix route $P_i'$ for $i$ with start position $p_i$ and start time $\tau_i$ that respects the current partial routes $\{P_j \mid j \in S\}$ and the given combinatorics $\rho$.
3. Let $E$ be the first transit segment on $P_i'$ such that $(j, i) \in \rho(E)$ and $E$ is not yet contained in the current prefix route $P_j$. Denote the beginning of the siding before $E$ on $P_i$ by $p'$.
4. Extend the prefix route $P_i$ by $P_i'$ until the position $p'$.

The location of $p'$ is well-defined only if it yields a proper extension, i.e., if there are at least two sidings between the end of $P_i$ and $E$. We say in this case that ship $i$ is *free*. This can be tested (or also administrated together with the route extensions of the other ships) in polynomial time.

It remains to be shown that Algorithm 1 is correct according to the requirements of Theorem 1.
Algorithm 1: Realizing the given combinatorics in our combinatorial relaxation.

### Lemma 1. Algorithm 1 produces an earliest arrival solution realizing \( \rho \) iff \( \rho \) is realizable for the instance \((C, S, R)\).

**Proof.** If Algorithm 1 returns a solution it is collision-free, earliest arrival, and realizes \( \rho \) by construction. Consequently, \( \rho \) is realizable.

For the converse direction assume that there is a realization \( \mathcal{P} \) of \( \rho \). W.l.o.g. \( \mathcal{P} \) is an earliest arrival solution. We prove the following loop invariant for Algorithm 1:

**Claim.** As long as \( S' \) is not empty there is at least one free ship \( i \in S' \) in each iteration and the constructed extension of \( P_i \) is equal to the corresponding part of \( i \)-th dynamic route in \( \mathcal{P} \).

This implies that the algorithm terminates with a complete collision-free dynamic route for each ship which finally equals the one of \( \mathcal{P} \).

The proof is by induction over the number of iterations. The base case is obvious. For the inductive step assume that the claim holds for all previous iterations and \( S' \neq \emptyset \). By contradiction, suppose that there was no free ship, i.e., each ship \( i \) with incomplete route has still an open predecessor \( j \) w.r.t. \( \rho \) on the next transit segment of its request after the current end of \( P_i \). When each ship has a predecessor blocking its route extension there must be a cycle in \( \rho \) since \( S' \) is finite. However, since \( \rho \) is realizable it is thus necessarily acyclic, a contradiction. Thus, there must be a free ship \( i \) in this iteration. By the inductive hypothesis, all prefix routes constructed in earlier iterations are equal to the corresponding parts in \( \mathcal{P} \). Hence, all prefix routes of the other ships preceding the current free ship on the transit segments of its next routing part within \( \mathcal{P} \) are already given and prevent ship \( i \) from traversing them too early. Hence, the corresponding unique earliest arrival route part constructed by the algorithm equals the part of the corresponding earliest arrival route of \( \mathcal{P} \).

For realizable combinatorial structure of a given instance the output of Algorithm 1 is well-defined; we thus conclude from the above induction hypothesis:

### Corollary 1. For any feasible solution to our combinatorial relaxation there exists a unique earliest arrival solution which can be constructed by Algorithm 1.

The local view on routing that we have adopted in this subsection will greatly help us in the following to incorporate feasibility constraints concerning passing and waiting in sidings when dealing with the original, unrelaxed problem.
4 Collision-free Routing for a single Ship

Assume that we are given collision-free dynamic routes \( P' \) for a subset of ships \( S' \subseteq S \). We would like to construct a dynamic route for a further ship \( i \in S \setminus S' \) which is collision-free and minimizes its waiting time w.r.t. the already given \( P' \). With siding conditions relaxed (Section 3.2) this insertion was easy via greedy routing. In the original setting, more work is needed for two reasons: First, if a conflict occurs, the siding directly in front of it may be fully occupied such that the waiting must take place at an earlier siding. It is thus not always possible to use the earliest available time to traverse a transit segment. Second, the routing algorithm must now take spatial decisions concerning collision-free waiting within sidings.

Our setting is related to the iterative routing of automated guided vehicles (AGVs) where transportation requests arise over time in a network. The routing algorithm in [12] answers the respective next request, avoiding collisions to previously routed AGVs already during the route computation. Despite significant differences concerning the practical contexts, we extend their algorithmic idea to plan ship \( i \) optimally with respect to the dynamic routes of \( P' \). It is worth noting that shortest path problems with time windows are widely studied as subproblem in the vehicle routing literature, see [6, 24] for overviews. The permanent labeling algorithm developed in [5] (see also [6]) is the basis for the algorithm of [12] which runs in polynomial time for the objective of finding a quickest path respecting time windows.

Like in Dijkstra’s algorithm [4, 7] we maintain labels representing arrival times of the ship. However, instead of referring to a single point in time, our labels correspond to feasible arrival time intervals. Further, for each arc, we store forbidden time windows during which it is not allowed for the ship to enter that arc in order to avoid collisions with conflicting ships. Together, this gives us an implicit time-expansion of the network and avoids the rather traditional way of guaranteeing collision-freeness by deleting arcs from a time-expanded network [10, 11, 22]. When exploring the outgoing arcs of a node the label intervals will be adapted according to the forbidden time windows. The collection of labels then covers all possibilities to traverse a transit segment, not only the earliest one. This resolves the first algorithmic issue stated in the introduction of this section.

Algorithm 2 gives an overview of our quickest path computation with time windows. A label \( \text{lab} = (a, [\text{epat, lpat}], \text{pred}, \ell) \) represents a set of collision-free dynamic routes for ship \( i \), from its start position \( s_i \) to the target node of arc \( a \), with an interval of earliest (epat) and latest (lpat) possible arrival times, and a reference to a predecessor label \( \text{pred} \) of \( \text{lab} \) on all represented routes. Using \( \text{pred} \), a route with arbitrary arrival time in \( [\text{epat, lpat}] \) can be reconstructed recursively. Since the length \( \ell \) of some arcs will be adapted dynamically by the algorithm, each label holds \( \ell \) as a further component. It is possible that a label interval degenerates to a single point in time. The precise construction of the network \( G(C) \) and the maintenance of forbidden time windows \( F(.) \) is described later.

Like in Dijkstra’s algorithm, the union of all labels is organized in a priority queue. We use a natural order \( \prec \) on the labels which is induced by the earliest possible arrival times \( \text{epat} \). For labels which are identical according to this order, preference is given to certain properties, see below. Controlled by this earliest-label-first order, the arcs are explored starting from labels corresponding to the arc that contains the requested start position until an arc containing the target position is reached. Two important sub-tasks distinguish the algorithm from Dijkstra’s classic (details follow):

1. The forbidden time windows are respected during the exploration of an arc resulting in a set of split labels. Furthermore, for arcs corresponding to a waiting track, the label
input: canal network $G(C)$, routing request $r_i \in R$, time windows $F(.)$
output: quickest path $P_i$ w.r.t. $F(.)$
1 enqueue start-labels for $s_i$ in priority queue $PQ$
2 while $PQ$ is not empty do
3 \quad cur := best label w.r.t. $\prec$ dequeued from $PQ$
4 \quad if $cur$ corresponds to $t_i$ then
5 \quad \quad return reconstructed dyn. route for $cur$ with earliest possible arrival time
6 \quad foreach successor arc $succ$ of label $cur$ do
7 \quad \quad foreach label $next$ resulting from propagation of $cur$ along $succ$ w.r.t. $F(.)$ do
8 \quad \quad \quad if $next$ is not dominated by an active label of $succ$ then
9 \quad \quad \quad \quad enqueue $next$ in $PQ$
10 \quad \quad \quad remember $next$ as an active label of $succ$
11 \quad \quad \quad delete active labels of $succ$ that are dominated by $next$ also from $PQ$

Algorithm 2: Calculate a quickest path for ship $i$ avoiding given forbidden time windows $F(.)$ within a canal network $G(C)$.

interval is expanded to represent the waiting. Together, this is called propagation.

2. Comparing two labels corresponding to one arc, we cannot always determine which one will produce no better solutions than the other. In this case, both must be remembered for further consideration. Otherwise, a dominated label can be deleted.

In the remainder of this section we specify the details to apply the algorithm in our canal setting: define an appropriate network, develop time windows guaranteeing collision-freeness, and describe the distinct steps of the algorithm. In particular, this answers the second algorithmic challenge concerning waiting decisions within sidings.

4.1 Network for collision-free Routing

The quickest path computation must take place in an appropriate network. Our use of forbidden time windows in the algorithm saves us from discretizing the time-horizon and from forming a time-expanded network. However, another complication arises from the spatial dimension when modeling sidings and their waiting tracks. Recall that each ship traverses the siding on the passage track, changes to the waiting track if it reaches the defined waiting position, and later continues its passage again on the passage track. It seems natural to partition the waiting track into a discrete set of “parking lots,” or waiting arcs, to and from which a ship can branch from arcs corresponding to the passage track. Unfortunately, this would cause several problems. With a fixed “reserved space” short ships would consume more waiting capacity than necessary. This would result in a loss of siding capacity. Answering to this by partitioning the waiting track in many short waiting arcs would also increase the number of labels and hence, the running time. Worse, a parking ship would then use several arcs, but the propagation along an arc should work locally, i.e., the reduction of the label interval to ensure collision-freeness should only depend on the time windows saved for that particular arc. For this, we would need to determine, after a route computation, which other arcs were affected when a single waiting arc is used by a ship for waiting and hence, on which arcs to store the resulting forbidden time interval. This could be handled when we assumed identical
ship dimensions as is the case for AGVs in [12]. In our setting, however, many individual ship lengths vary from 50 to 250 meters. Finally, no partition of the waiting track could not ensure that a ship waits at most once per siding. For these reasons, we refrain from statically partitioning the waiting tracks, i.e., we do not use a spatial discretization at all. Instead, we let our algorithm dynamically consider waiting positions as necessary.

![Figure 9: Network $G(C)$ representing canal $C$. In a siding, different types of arcs reflect time/space before (type $\alpha$), while (type $\beta$), and after (type $\gamma$) waiting.](image)

Figure 9 displays how we translate a canal into a network. A node is introduced on the boundary points of each segment. For each segment, these two nodes are connected by an up- and a downstream arc, respectively. In order to model the possibility to wait on the waiting track, we introduce a further node per direction within each siding. It allows the ship to interrupt the passage of the siding for waiting and continue the passage later on. The waiting track itself is represented by a loop adjacent to this node. Hence, there are three arcs per siding and direction, one of type $\alpha$ for the passage before waiting, one of type $\beta$ for waiting on the waiting track, and one of type $\gamma$ to continue the passage after waiting. In contrast to the nodes at the segment boundaries the node representing waiting in a siding has no explicit embedding: The waiting arc does not correspond to a fixed waiting position but will hold information about all ships of $P'$ waiting within that siding. As part of the propagation process along arc $\alpha$ all necessary but not more positions for the waiting node will be checked depending on this information. Given a canal $C$, we denote this network as $G(C)$. All the potential complexity of this network—we neither have time nor space discretization—is avoided by entirely shifting the information management to the algorithm.

### 4.2 Forbidden Time Windows

Consider a transit segment $E \in E \setminus T$. As suggested by Equation (9) the dynamic route of ship $j \in S'$ (in conflict with the new ship $i$ on $E$) defines an upper bound $u_j := d_{jE} - \Delta(i,j,E)$ on the visit time of $i$ on $E$ before the passage of ship $j$. Similarly, a lower bound $\ell_j := d_{jE} + \Delta(j,i,E)$ for visiting $E$ after $j$ arises. This results in a time window $F_j := [u_j, \ell_j]$ implied by the dynamic route of $j$ during which it is forbidden for ship $i$ to enter $E$, i.e., we have to ensure that $d_{iE} \notin F_j$. As before, the interval is with respect to the start position of segment $E$ to have a reference point independent of $s_i$. In this sense, we can determine for each arc and each dynamic route in $P'$ time windows that restrict the use of this arc to ensure a collision-free dynamic route for $i$.

Now consider a siding $[x, y] := T \in T$. A ship that waits in $T$ does not block the passage track of $T$. Therefore the passage of that track before waiting and the passage after waiting must be considered separately. Consider a ship $j \in S'$ with given dynamic route that waits at position $p_j \in T$ and has a conflict with $i$. If ship $i$'s waiting position $p_i$ is, say, in front of $p_j$, we must ensure that the passage of ship $i$ before its waiting does not collide with the passage of $j$ before its waiting and furthermore with the passage of $j$ after its waiting. Hence, depending
on the considered waiting position \( p_i \) ship \( j \) can induce two forbidden time windows for the corresponding arc of type \( \alpha \) or two for the corresponding arc of type \( \gamma \). For more details confer Figure 10. Additionally, the waiting ship \( j \) produces a forbidden time window on the waiting arc of \( T \) (type \( \beta \)) for the interval \( \tau_j(p_j) \) if the waiting positions \( p_j \) and \( p_i \) have a distance smaller than \( h_i + h_j \).

![Diagram showing two scenarios for forbidden time windows.](image)

Figure 10: A given dynamic route of ship \( j \) defines a set of time windows in sidings. If ship \( i \) is supposed to wait behind ship \( j \) (part (a)) there are two time windows where \( i \) is not allowed to enter \( T \). Otherwise (part (b)) ship \( j \) induces two time intervals where ship \( i \) is not allowed to continue its passage after waiting. To determine the boundaries of the forbidden time windows we make use of \( \Delta(\cdot) \) as above. Since it is possible that ship \( i \) and ship \( j \) have different velocities, we must restrict the domains of \( i \) and \( j \) valid for the calculation of \( \Delta(\cdot) \) for the distinct cases (indicated by the light areas): \( F_1^\alpha \) corresponds to \([x,p_i]\) and \([x,p_j]\), \( F_2^\alpha \) to \([x,p_i]\) and \([p_j,y]\), \( F_1^\gamma \) to \([p_i,y]\) and \([x,p_j]\), and finally \( F_2^\gamma \) to \([p_i,y]\) and \([p_j,y]\) if the respective intervals are not disjoint (cf. Figure 8). If the start- or target position of \( i \) or \( j \) is situated within this siding, the request intervals must be further reduced appropriately. For reasons of clarity we located the time windows of the arcs of type \( \gamma \) in this picture w.r.t. the end of \( T \). Note that we actually define them translated to the start of \( T \) such that they are compatible with all other definitions.

To summarize, the set of dynamic routes \( \mathcal{P}' \) yields sets of forbidden time windows \( \mathcal{F}(a,i,p_i) \) for siding arcs \( a \), and sets of forbidden time windows \( \mathcal{F}(a,i) \) for transit arcs \( a \). These restrict the use of \( a \) for ship \( i \) as described, possibly depending on the given waiting position \( p_i \).

As already mentioned, we want to have fast local access to the set of time windows of an arc \( a \) during the propagation along \( a \). To this end, each arc holds for each given dynamic route \( P_j \in \mathcal{P}' \) the respective local information to define a time window (the visit time of the segment, possibly the waiting position within the siding, and a link to the properties of the corresponding ship \( j \)). Since each time window additionally depends on properties of ship \( i \) it can be calculated in constant time.

### 4.3 Routing Details for the Canal

We now explain the single steps of Algorithm 2 in more detail.
Propagation. The core task of propagating a label \textit{cur} along an arc \textit{succ} with respect to \( F(.) \) (Line 7 of Algorithm 2) is the splitting of a label interval corresponding to the given forbidden time windows into a set of new labels for arc \textit{succ} guaranteeing collision-free dynamic routes. This splitting is described in Algorithm 3 the input of which must be chosen according to the type of the considered arc \textit{succ}.

```
Algorithm 3: Splitting a predecessor label along an arc \textit{succ} into a set of feasible labels for \textit{succ}.

\textbf{input:} arc \textit{succ}, label \textit{cur}, time value \textit{newend}, union of time intervals \( F \), label length \( \ell \), transit time \( \tau \)

\textbf{output:} Set of feasible labels for \textit{succ} of length \( \ell \) with \textit{cur} as predecessor

1 \( \mathcal{I} := [\text{epat}(\textit{cur}), \text{newend}] \setminus F \) // open time windows
2 \( \mathcal{L} := \emptyset \)
3 \textbf{foreach} maximal interval \([\text{epat}', \text{lpat}']\) of \( \mathcal{I} \) with \( \text{epat}' \leq \text{lpat}(\textit{cur}) \) do
4 \( \mathcal{L} := \mathcal{L} \cup (\text{succ}, [\text{epat}' + \tau, \text{lpat}' + \tau], \textit{cur}, \ell) \)
5 \textbf{return} \( \mathcal{L} \)
```

If \textit{succ} corresponds to a transit segment \( E \), the algorithm is called with the following arguments: \( \text{newend} := \text{lpat}(\textit{cur}), F \) as union of \( F(\text{succ}, i) \), label length \( |E| \), and transit time \( \tau_iE \). Since \( F \) contains each point in time at which entering of \( E \) is prohibited for ship \( i \), each time value of \( \mathcal{I} \) is a feasible entering time for \( i \). Since the new labels correspond to the target node of \textit{succ}, the time intervals must be translated to this position.

If \textit{succ} of type \( \alpha \) for a siding \( T \), the splitting must be done several times, namely for each interesting waiting position \( p_i \). Each iteration calls Algorithm 3 with \( \text{newend} := \text{lpat}(\textit{cur}), F := \text{union of } F(\text{succ}, i, p_i), \) label length \( := p_i \), and the resulting transit time for this length as input. To determine the set of all interesting waiting positions, each maximal time interval with invariant waiting conditions within \( T \) from the earliest arrival of \textit{cur} is considered. In this, each free waiting area will be filled with waiting positions respecting the horizontal parking distances with \( h_i \) as step size from the end of the siding to its beginning until one waiting position yields a label who’s interval would not be reduced by the splitting process on any of the three siding arcs. All further waiting positions would not yield further possible arrival times at the end of the siding and hence, would not yield a better solution. This reduces the number of constructed labels often drastically and is important for an acceptable running time.

The input value \textit{newend} becomes important, if \textit{succ} is an arc of type \( \beta \) for siding \( T \). In this case, the waiting will be modeled by an extension of the interval to \( \text{newend} := \infty \). After this extension it will be split by \( F \) as union of \( F(\text{succ}, i, \ell(\textit{cur})) \) since \( \ell(\textit{cur}) \) is the tested waiting position. To provide this information also for the next arc, the label length is also set to \( \ell(\textit{cur}) \) (and not 0). But the transit time is 0.

Finally, we describe the propagation for the case that \textit{succ} corresponds to an arc of type \( \gamma \) for siding \( T \). In this case, label \textit{cur} can correspond to arcs of both types \( \alpha \) and \( \beta \). The start node of \textit{succ} then has the corresponding position and hence, the given label length must be \( |T| - \ell(\textit{cur}) \) and the transit time, respectively. Furthermore, each time window of \( F(a, i, \ell(\textit{cur})) \) must be translated by \( \ell(\textit{cur}) \) w.r.t. the velocity of \( i \) for the construction of \( F \). The value \textit{newend} is again set to \( \text{lpat}(\textit{cur}) \).
**Dominance.** A label lab is said to dominate another label lab′ if both represent the same position in the canal, and lab′ cannot lead to any better routes than lab. Dominated labels can be deleted. More precisely, label lab dominates lab′ if and only if both correspond to the same arc a, both correspond to the same length ℓ if a is of type α or β, and

\[ \text{lab} \preceq \text{lab′} \quad \text{and} \quad [epat(\text{lab′}), lpat(\text{lab′})] \subseteq [epat(\text{lab}), lpat(\text{lab})]. \]

Testing the dominance of labels (Lines 8 and 11 of Algorithm 2) helps to reduce the number of labels that must be propagated without losing any dynamic route of minimum waiting time.

**Resulting Dynamic Route.** The first constructed label that corresponds to an arc containing the target position of ship i (Line 5 of Algorithm 2) represents a set of dynamic routes with smallest possible arrival time and hence, with minimum waiting time. It may still be open in which of the sidings in front of an occurred conflict to spend the waiting time. Each route in the set can be constructed recursively from the last label in reverse order. Hence, we have the freedom to choose among these optimal dynamic routes and we decide to let waiting always take place within the latest possible siding. Also, the ordering \( \prec \) of the priority queue is defined such that late waiting positions within each siding are preferred. Consequently, in the case of relaxed siding conditions, Algorithm 2 produces the same earliest arrival solutions as greedy routing. After the reconstruction of the dynamic route of i we update the corresponding information on each arc for future calculations of time windows.

When there is not enough capacity available for waiting within the sidings, the priority queue runs out of labels before a label corresponding to the target position is reached. In that case we conclude that a dynamic route for ship i does not exist.

**Running Time.** Determining the running time of the routing procedure includes to take care of the number of waiting positions that need to be tested. This number is strongly related to the maximum ratio \( \rho_{hi} \) of siding length and horizontal distance of ship i. With some extension of the argumentation in [12], it can be shown that the running time is polynomial in the maximum number of time windows \( \eta \) per arc, the number of segments, and \( \rho_{hi} \). Note that \( \eta \) is bounded by \( 2n \) and is often much smaller. For the real world instances coming from the Kiel Canal each ratio \( \rho_{hi} \) is bounded by a constant.

**Knapsack Relaxation.** Assuming relaxed siding conditions, a solution can be realized from the given combinatorics via greedy routing (Theorem 1). When we additionally impose the siding capacity as a knapsack constraint, it might be impossible for each ship to wait directly in front of a conflict. Thus, greedy routing may produce infeasible solutions. Even so, Theorem 1 can be extended to this stronger knapsack relaxation (which is relevant in the case of all identical ship dimensions) when using collision-free routing instead of greedy routing: only time windows and propagation on arcs of type β need to be slightly adapted.

**Corollary 2.** For the knapsack relaxation, there is a polynomial time algorithm using collision-free routing that, given an instance \((C, S, R)\) and a combinatorial structure \(\rho\) on \(S\), either creates a feasible earliest arrival solution \(P\) realizing \(\rho\) or proves that no realization exists.
5 A Heuristic for the STCP

5.1 Construction of Solutions by Sequential Routing

We use a natural idea [12] to construct an initial solution from scratch: We iteratively route ships through the canal, one after another, using collision-free routing. We sort ships by non-decreasing times they (would have) entered the canal (i.e., the translation of each release date from the start position to the canal boundary, if needed). This is a canonical ordering to route ships one by one, but it is somewhat arbitrary, and there may be better orderings which give better overall solutions. However, the question for a best ordering of such a sequential routing is void in view of the following lemma.

Lemma 2. Any sequential routing procedure can yield arbitrarily bad solutions, even if all horizontal distances between ships are zero.

Proof. Figure 11 schematically sketches an instance where in each optimal solution each ship has to wait for a short time of, say, at most $\varepsilon$. Such a solution contains a cyclic waiting pattern which cannot be achieved via any sequential routing procedure: at least the ship which is routed first will not wait anywhere. However, such a ship causes another ship to wait for at least $K \gg \varepsilon$. With $\varepsilon \to 0$, one cannot avoid an arbitrarily large deviation from the optimum when routing sequentially.

5.2 Improving Schedules by Local Search on the Combinatorics

We have seen already earlier that decisions about who is waiting for whom and where constitute the core difficulty of our problem. The most important insight from Lemma 2 is that, regardless of the ordering of ships, sequential routing is limited in the structure of schedules that it is able to produce. Actually, it does not actively produce a schedule at all, i.e., the combinatorics of the solution comes as a necessary by-product. We therefore need to find ways to produce different, in a sense richer combinatorial structures, and we need to be able to deal with this in conflict-free routing. We address the second item first.

Algorithm 1b. A given combinatorial structure may be impossible to realize because of the siding capacities. The following modification of Algorithm 1 succeeds in producing a
realization if this is possible. However, we allow to alter or even eliminate some scheduling
decisions if that should be necessary. When in Algorithm 1, Line 4, a route cannot be extended
for ship $i$ due to insufficient space for waiting in the preferred siding, we backtrack: Try the
route extension from one siding earlier, dismissing the already fixed onwards part of the
extension. This is iterated until we find a dynamic route for ship $i$ or no earlier siding is
available. In the latter case, no dynamic route can be assigned to ship $i$. Note that this
backtracking does not touch other ships: a ship which originally succeeded ship $i$ along a re-
considered transit segment may change sequence with ship $i$. We therefore cannot guarantee
that the route extension of Line 4 respects a given combinatorial structure in general. However,
barring the mentioned complications it does. The backtracking version of Algorithm 1 is called
Algorithm 1b.

**Combinatorial Neighborhood.** We would now like to improve existing solutions by local
search, as summarized in Algorithm 4. Instead of working with our geometric encoding of an
incumbent solution $P$ directly, we define a neighborhood on the induced combinatorics $\rho(P)$,
slightly abusing notation. This fits well with our conclusions from Lemma 2 since it enables
us to produce a much larger variety of schedules in the first place. A neighbor of $\rho(P)$ is
constructed by altering scheduling decisions on a single segment. Figure 12 motivates how
this is done: every ship $i$ waiting in a siding could switch precedence on the onward transit
segment $E$ with (some of) the $k$ conflicting ships it is waiting for. Let $j_1, \ldots, j_k$ be the ordering
of these ships on $E$. All of them currently precede $i$ on $E$. Every re-ordering \{i, j_1, \ldots, j_k\},
\{j_1, i, j_2, \ldots, j_k\}, \ldots, \{j_1, \ldots, j_{k-1}, i, j_k\} induces new precedences $\rho'(E)$ on $E$. The union
of all these, over all waiting ships in all sidings, describes the neighborhood of $\rho(P)$. When
several ships wait in a siding for the same set of conflicting ships, it makes sense to insert all
of them simultaneously in $j_1, \ldots, j_k$. This is done in our implementation.

![Waiting ship induces several neighbors](image1)

Figure 12: A waiting ship induces several neighbors in (a); a new incumbent solution in (b). This also illustrates how our local estimate of a neighbor is computed. Assume that ship 4
is not in conflict with ship 8. Each dashed line in (a) represents an opportunity for ship 8 to
continue its journey. Realizing one of these would require swapping the scheduling relation for
all succeeding ships on this transit segment. The most promising opportunity as in (b) yields
a large benefit for ship 8 and three small increased waiting times for the ships 5, 6 and 7.

The described neighborhood may contain combinatorial structures that are not realizable,
not even in the combinatorial relaxation of Section 3.1 that ignores siding capacities. Theorem 1 ensures that we detect this defect, and an adapted Algorithm 1 can repair it. In Line 7
of Algorithm 4 we ensure that there are free ships as long as there are ships with an incomplete route by heuristically deleting a relation on a cyclic dependency (similar to Figure 11a) when this is detected.

**Local Estimates of Neighbors.** Representing a solution only by its combinatorics adequately reflects our wish to improve scheduling decisions. At the same time, however, this also entails complications like the problem of how to evaluate a neighbor \( \rho' \): It would be too expensive to use Algorithm 1b only to calculate the waiting time of a solution produced from \( \rho' \). As a remedy we only use a simple estimate, see again Figure 12: We compare the decreased waiting time (benefit) of a ship, or a group of ships, with the increased waiting time (loss) of the conflicting ships we delay. This is a very myopic analysis ignoring that reversing scheduling decisions locally can have global effects: because of siding capacities, waiting time can be saved or produced in a siding distant from the altered scheduling. Only neighbors with best local estimates are actually evaluated by trying to realize \( \rho' \) via Algorithm 1b. Recall that we cannot guarantee that all altered scheduling decisions are respected, and that it may even happen that a ship cannot be routed at all. A solution with best actually realized improvement becomes the next incumbent. In fact, in order to escape local optima we allow worsening iterations. A parameter limits the total number of such failures of improvement. We avoid cycling by tabooing bad incumbents.

```
input: canal C, requests R for ships S
output: solution P
1 construct initial solution P by sequential routing
2 repeat
  3 \( \rho(P) := \) combinatorics induced by P
  4 estimate benefits/losses of all neighbors of \( \rho(P) \)
  5 \( N := \) subset of neighbors with best local estimates
  6 foreach neighbor \( \rho' \in N \) do
  7     ensure/repair (combinatorial) realizability of \( \rho' \) using Theorem 1
  8     construct a solution from \( \rho' \) using Algorithm 1b
  9     \( P := \) best found solution for candidates in \( N \)
21 until there is not enough improvement
11 return globally best found solution
```

**Algorithm 4:** The general scheme of our local search on the combinatorics

**Heuristically Encouraging Certain Properties of a Solution.** For reasons of practical acceptance of our solutions we need to take care of a few desired properties which have not been algorithmically addressed so far. We therefore do not evaluate solutions only by their total waiting times for several reasons: (a) Algorithm 1b may leave ships unrouted which nominally decreases waiting time; (b) very large individual waiting times of single ships may decrease that of many other ships, however in practice, waiting in a siding (and also the total waiting time per ship) should not exceed certain soft limits; (c) a large individual waiting time of a single ship counts as much as the sum of short waiting times for several ships; (d) it is a good idea to give priority to slow ships as they generally produce more scheduling complications in particular for opposed ships. These problems can be lessened by assigning individual weights
to ships and by penalizing large individual waiting times. In our implementation this is done by replacing the objective of total waiting time by $\sum_{i \in S} (a_i w_i)^f$ with priorities $a_i$ and $f > 1$. This applies to both, estimating a neighbor in Line 4 and choosing a best new incumbent in Line 9 of Algorithm 4.

5.3 Rolling Horizon

Finally, we embed the local search in a rolling horizon framework, i.e., we consecutively plan for shorter, overlapping time horizons. There are two good reasons for doing so—an algorithmic and a conceptual one. Algorithmically, this supports the local search as more neighbors can be evaluated when dealing with a smaller instance i.e., a shorter time horizon. Conceptually, we more realistically emulate the planning process at the Kiel Canal where requests arrive over time, i.e., in an online manner.

We denote the horizon length by $T_\Delta$, the horizon start time by $T_0$, and the step size by $T_\delta \leq T_\Delta$. In each iteration we consider only a partial instance of those ships with release time between $T_0$ and the horizon end time $T_0 + T_\Delta$. Note that it is important to construct each dynamic route until the actual target position of the ship even if it will arrive there after $T_0 + T_\Delta$. After constructing a solution for this partial instance, we declare everything before the horizon end time as fixed and define the intersection point of each calculated dynamic route with the corresponding time axis as new start position and release time. Now, $T_0$ increments to $T_0 + T_\delta$ and the process repeats until all released ships have been considered.

Fixing everything before the horizon start time bears the risk that for some ships no dynamic route can be found since it can not be adapted if siding capacities are exceeded afterwards. Besides the construction of dynamic routes to the actual target position beyond the scope of the horizon length we also transmit the already taken scheduling decisions $\rho$ from one time horizon to the next to decrease this risk.

Experiments in Section 6.1 will show that this results in better objective values, running times and even in fewer ships without assigned dynamic route compared to plain local search.

6 Computational Study

GPS position data are collected permanently for every vessel in the canal. The entire data (of about 43,000 ships) of actually traveled itineraries for the busy year 2007 were made available to us. From these we generated 365 instances, one for each day with a planning horizon of 24 hours. They contain 185 requests on average (minimum 83, maximum 247). Algorithms were implemented in Java. All computations for the following study were performed on a commodity desktop PC running Linux.

When evaluating the quality of a solution we refer to average waiting time per ship. The (penalty) waiting time of a ship that could not be routed by an algorithm is set to 2 hours. All calculations are exact to the second (typically not practicable in time-discretized approaches). Figure 13 shows the two visualization options we offer in our tool to validate our solutions. Both greatly helped in fine tuning and granting approval by the expert planners.

6.1 Algorithmic Components

We stressed the interplay of geometry and combinatorics several times. Our first suite of experiments aims at demonstrating that this problem understanding also pays off compu-
tationally. We evaluate the “purely geometric” approach of sequential routing (Section 5.1, denoted SeqR); sequential routing integrated with re-scheduling decisions in a local search on the combinatorics (Section 5.2, denoted LS-240); and the rolling horizon heuristic which additionally reflects the problem’s online character (Section 5.3, denoted RH-1). The numbers refer to the following parameter settings. LS-240 is allowed a maximal number of 240 worsening steps within the local search over the full 24-hour planning horizon. In RH-1 the horizon length is $T_\Delta := 2$ hours and the step size is $T_\delta := 1$ hour. As the local search acts only on horizons of an hour, we allow a maximum number of 10 (i.e, $240/24$) worsening steps within the local search to make LS-240 comparable to RH-1 in that respect. Furthermore, we greedily evaluate only one promising neighbor-candidate within an improvement step, i.e., $|\mathcal{N}| = 1$ in Algorithm 4. The variant with $|\mathcal{N}| = 3$ is denoted by RH-3.

We present four pairwise comparisons of algorithms. In each comparison we compute, for each instance, the ratio of average waiting times obtained by the respective two algorithms and give the whole distribution of ratios as a box-and-whiskers plot in Figure 14. The box marks the range of the mid 50% of the ratios where the central line indicates the median. The whiskers mark the range of 95% and the smallest and largest 2.5% are considered as outliers, marked by circles. The average (marked with an asterisk *) and the standard deviation are given as information.

Both, the integration of routing and scheduling (LS-240) and the rolling horizon heuristic (RH-1) significantly improve over sequential routing (SeqR) alone. It is remarkable that the local search improvement performs considerably better when it works in small chunks “over time” instead of more globally on the whole day, as shown by the comparison RH-1/LS-240. The last comparison RH-3/RH-1 indicates that it does not pay much to evaluate neighbor-candidates in a way more elaborate than greedily. The most important qualitative conclusion is that taking care of scheduling decisions in addition to sequential routing is imperative for obtaining satisfactory solutions.

Table 1 summarizes two further aspects: the number of ships for which no dynamic route could be found (a little more than one ship per instance) and computation times (which are generally very low). RH-1 yields the best combination of quality and running time.

### 6.2 Combinatorial Relaxation

In Section 3.1 we discussed a combinatorial relaxation that ignores siding constraints. We will
use its MIP formulation (1)–(8) to estimate the quality of solutions obtained by our rolling horizon heuristic RH-1. This MIP is computationally very demanding: a still unsolved instance (called shipsched) with 87 ships belongs to the challenge benchmark set of MIPLIB2010 [15]. We therefore prepared smaller instances comparable to one time horizon in RH-1 as follows. Start from a given solution for a whole day; filter all ships that are present or enter the canal during a given time horizon of two hours; and further reduce the number of ships to at most 35 by random removal, biased to prefer a balance between upstream and downstream ships and to keep ships satisfying a minimum distance value between start and target position.

Figure 15 shows results for 21 instances. Label “MIP” marks the value of the best known feasible integer solution of MIP (1)–(8). For instances not solved to optimality we additionally state the largest known lower bound indicated with the label “LB.” The objective values labeled by “RH-1” refer to the rolling horizon heuristic. To add another level of comparability we also consider a version of the rolling horizon heuristic where siding constraints are relaxed as in the MIP; the corresponding label is “relaxRH-1.” As an information we state the computation times needed by the state-of-the-art MIP solver CPLEX (version 12.3). They indicate that the complexity of instances of 35 ships varies from very easily solvable to intractable.

The average difference of waiting times between LB/MIP and relaxRH-1 is slightly above 5
Figure 15: Results for 21 small instances with at most 35 ships over a 2h horizon. The columns indicate the respective average waiting times per instance for the rolling horizon heuristic (RH-1), this heuristic with relaxed siding conditions (relaxRH-1), the value of the best known integer solution of MIP (1)–(8), and (if not identical to MIP) the largest known lower bound (LB). MIP/LB constitute lower bounds on the optimum. A dagger † indicates that RH-1 was not able to find a dynamic route for all ships of the corresponding instance due to capacity problems within sidings. The CPLEX computation time to solve the MIP is stated in CPU minutes, or “mem” when the computation hit the memory limit.

minutes (maximum is 18 minutes). These gaps increase a bit if the siding conditions must be respected (average below 7 minutes, maximum below 22; ignoring the two instances marked with *). One might have expected a larger increase. There are even two instances where RH-1 produces less waiting time than relaxRH-1. In these cases, siding capacity was not critical but induced different decisions within the local search process.

Figure 16: Comparison as in Figure 14 (all instances of 2007): our rolling horizon heuristic (RH-1) vs. its variant with relaxed siding constraints (relaxRH-1).

Returning to the 24-hour instances, this effect is still visible but remains exceptional, see Figure 16. On average, RH-1 loses only 8% when compared to relaxRH-1, but this increases drastically (to about 30%) for the prognosticated traffic demand. This shows the importance of modeling the siding constraints in detail.
6.3 GPS Data Realized

We finally evaluate the practical usefulness of our algorithms. The GPS data contain actual entry/exit times at sidings for each ship. Figure 17(a) shows the linear interpolation between them. Small circles indicate infeasibilities of such a solution: large ships pass each other (slightly) outside sidings; velocities do not exactly match the actual ones (some ships exceed the speed limit in reality, others move below full speed, in particular when waiting in front of the lock chambers at the canal boundaries); also constraints in sidings may be violated.

We deduce a combinatorial structure from this solution, resorting to the most plausible scheduling decision in case of ambiguities, and use Algorithm 1 to construct a routing. The latter may not exactly realize the original solutions since these need not be earliest arrival ones in general. However, we consider this proceeding a closest-to-reality re-construction of actual itineraries which at the same time fit the precise feasibility definitions. The result is called GPS data realized, GPS-re.

Figure 17: Actually traveled itineraries as per GPS data with infeasibilities on the left (a) and a feasible solution derived using Algorithm 1b on the right (b).

Figure 18: Distribution of the instance-wise improvement w.r.t. average waiting time of the rolling horizon heuristic (RH-1) over realized GPS data (GPS-re).

Figure 18 shows the improvement of the rolling horizon heuristic over manually planned solutions. Again, precise values should be taken with care, but a tenor is clearly visible. More interesting is the behavior pointed out in Figure 19. Instances are sorted according to average waiting time which is considered a proxy for the planning complexity. Instances which are harder under this measure in reality are harder for our heuristic, too. This similarity brings us in a position to reliably emulate predicted traffic under different canal enlargement options.
7 Conclusions and Perspectives

We discussed the complex problem of ship traffic control at the Kiel Canal. We integrated algorithmic ideas from two important related applications, train scheduling on a single-track network and collision-free routing of automated guided vehicles. This unified view of routing and scheduling allowed us to find a blend between simultaneous (global) and sequential (local) solution approaches to allot scarce network resources to a fleet of vehicles in a collision-free manner.

We developed a fast heuristic, with an average running time of less than two minutes, which yields solutions that are approved by the expert planners. Instance-dependent lower bounds prove the quality of the achieved objective function values which significantly improve upon manual plans. Much more importantly, we model the practical context in such a high level of detail that the resulting tool perfectly reflects the effects of enlargement options at the Kiel Canal. This enabled the officials to evaluate the different options under ship traffic predicted for the year 2025, and to base their decisions on the simulation results.

Even though it came as a side effect of the study, we lay the ground for a computer aided traffic control. In fact, our planning in rolling horizons integrates with a heuristic [19] that schedules the locking process at each boundary of the Kiel Canal since entering, passing, and exiting the canal are interdependent. The overall system may support the expert planners during several potentially difficult years of construction work. Moreover, it was considered to use the tool for deciding about the schedule of the construction work itself: Different orders of construction and the selection of different excavating machines (several of which significantly hinder regular traffic) directly influence the traffic flow under scarce resources.

We are confident that our ideas can be extended to other application areas like train timetabling and collision-free routing, also in more general networks.
Acknowledgments. We are grateful to our cooperation partner “Planungsgruppe für den Ausbau des NOK” of the Waterways and Shipping Administration, in particular Martin Abratis, Ulrich Bösl, Martin Bröcker, and Sönke Meesenburg for their openness to mathematical methods, for their efforts and patience at intense communication processes, and in general for a very delightful cooperation. We thank Felix G. König for stimulating discussions about algorithmic ideas and intensive support within this project. We acknowledge support by the German Research Foundation (DFG) which enabled our production of a series of educational short movies for high school students, based on this project [17].

References


