OR Spectrum (2020) 42:995–1021
https://doi.org/10.1007/s00291-020-00599-5

REGULAR ARTICLE

Robust spotter scheduling in trailer yards

Giorgi Tadumadze[1] · Nils Boysen[2] · Simon Emde[3]

Received: 4 August 2019 / Accepted: 23 July 2020 / Published online: 17 August 2020

Abstract

Spotters (also denoted as switchers) are specialized terminal tractors, which are dedicated to the rapid maneuvering of semitrailers between parking lot and dock doors in large trailer yards. This paper is dedicated to spotter scheduling, i.e., the assignment of predefined trailer movements to a given fleet of spotters. The limited number of dock doors for loading and unloading is often the scarce resource during trailer processing, so that idle time of the bottleneck, e.g., caused by unforeseen delay in the yard, is to be avoided. In this setting, we aim to insert time buffers between any pair of subsequent jobs assigned to the same spotter, so that small delays are not propagated and subsequent jobs can still be executed in a timely manner. We formalize two versions of the resulting robust spotter scheduling problem and provide efficient algorithms for finding optimal solutions in polynomial time. Furthermore, we simulate delays during the execution of spotter schedules and show that the right robustness objective can greatly improve yard performance.

Keywords Yard operations · Truck scheduling · Terminal tractor scheduling · Robustness

✉ Nils Boysen
nils.boysen@uni-jena.de
http://www.om.uni-jena.de

Giorgi Tadumadze
giorgi.tadumadze@tu-darmstadt.de
http://www.or.wi.tu-darmstadt.de

Simon Emde
siem@econ.au.dk
https://pure.au.dk/portal/en/siem@econ.au.dk

[1] Fachgebiet Management Science/Operations Research, Technische Universität Darmstadt, Hochschulstraße 1, 64289 Darmstadt, Germany

[2] Lehrstuhl für Operations Management, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, 07743 Jena, Germany

[3] Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, 8210 Aarhus V, Denmark

1 Introduction

Increasing freight traffic in many regions of the world (e.g., in Europe Statista 2018) does not only burden the edges (streets) of road networks but also many nodes. Examples of huge terminals where up to several hundred trucks are to be loaded and unloaded each day are cross docks (Ladier and Alpan 2016), automobile plants (Battini et al. 2013), distribution centers of food retailers (Bodnar et al. 2015), freight airports (Ou et al. 2010), and hub terminals in the postal service industry (Boysen et al. 2017). In many of these nodes, especially during peak hours, considerable waiting times for an (un-)loading of trucks occur. The large German transport cooperative Elvis with 10,643 trucks, for instance, reports that their average daily waiting times of 3.5 h per truck accumulate to yearly waiting costs of about €400 million (VRS 2012). In an empirical study, 45.9% of the surveyed German truck drivers quantify their average waiting time per stop to exceed 1 h (VRS 2011). Existing ideas on how to resolve this problem mainly address the demand side. Novel software solutions propagate an internet-based booking of time windows, so that truck arrivals can be spread out over the day (Descartes 2019), and the current scientific research aims at a coordination via auction mechanisms (Karaenke et al. 2019). We, however, address the internal processes and show that robust schedules for the movement of semitrailers within yards also have the potential to considerably reduce waiting times.

1.1 Trailer yard operations and literature review

An important decision to make in the aforementioned nodes is to either let the trucks transporting semitrailers to and from the terminal directly approach the dock doors or to apply dedicated spotters for the intra-terminal trailer movement. Spotters (also denoted as switchers Yano et al. 1998 or terminal tractors Berghman and Leus 2015) are specialized truck tractors which are exclusively applied for the rapid maneuvering of semitrailers between their parking positions and the dock doors where trailers are loaded and unloaded. A spotter is depicted in Fig. 1a.

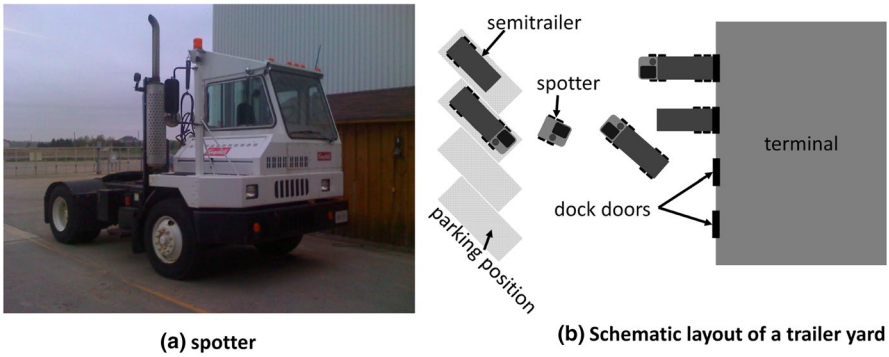


Fig. 1 Spotter (**a** is published under the Creative Commons Attribution Share Alike 3.0 Unported License. The author of the picture is Exit2DOS2000)

Applying extra spotters allows accelerated maneuvering and earlier release of the trucks. Spotters are more agile than conventional trucks, specifically designed for a rapid (un-)coupling process of semitrailers, and due to learning effects, their drivers are much better trained in backing into a dock. A terminal manager of a large German trailer yard we visited told us that spotters cut transportation time in the yard at least in half. Moreover, spotters allow for a decoupling of trailer processing at the dock doors and trailer transportation (beyond the yard). Inbound trucks can leave their trailer on some open parking space in the trailer yard and can directly head toward their successive appointment. They need not wait for the processing time window of their trailer, which may considerably improve truck utilization. On the outbound side, dock doors are not blocked by already loaded trailers waiting for a delayed truck to pick them up. On the downside, spotters incur additional investment and operational costs.

When applying spotters for trailer movement, the yard processes are organized as follows. On the inbound side, an incoming truck reaching a trailer yard is registered at the terminal gate and appointed an open parking space in the parking lot of the yard. Figure 1b depicts the schematic layout of a trailer yard. Once the trailer is positioned and uncoupled, the truck can directly leave the terminal. The arrival of the loaded trailer is registered in the yard's information system, e.g., by a yard operator controlling the parking lot with video surveillance. Then, the trailer is assigned a dock door and a processing time window. In the literature, the decision on the *where* and *when* of (un-)loading a given set of trailers is denoted as truck scheduling. A survey on the numerous solution procedures for this decision task is provided by Boysen and Fliedner (2010). Once such a schedule is available, loaded trailers need to be delivered in a timely manner to their dedicated docks by spotters. The spotter drivers are informed either via radio communication by the yard operator or via a display mounted on the dashboard. Once docked, trailers are unloaded, e.g., by forklifts or directly onto a telescopic conveyor, while the spotter heads to the next trailer on its list. When a trailer is completely unloaded, the status is transferred to the information system and a new transport request is generated as well as assigned to a spotter, which is to return the empty trailer to a parking space. On the outbound side, these steps are analogously executed in reverse order.

This paper is dedicated to spotter scheduling. A given set of trailer movements either from the parking lot to some dock door (i.e., loaded inbound trailers or empty outbound trailers) or vice versa (i.e., empty inbound trailers or loaded outbound trailers) are to be assigned to a given fleet of spotters. Truck scheduling problems from the literature almost without exceptions typically ignore the workforce scheduling aspect. However, Tadumadze et al. (2019) propose integrated truck and workforce scheduling problem but their workforce is dedicated to the intra-terminal operations (i.e., unloading goods from trailers at dock door) while spotter scheduling treats operations in the yard, i.e., outside of the terminal. The only other papers treating spotter scheduling are the ones by Berghman et al. (2014) and Berghman and Leus (2015). Both papers integrate truck scheduling, i.e., the decision on the where and when of truck processing, with spotter scheduling and model the complete process as a three-stage flexible flow shop problem. On the first stage, spotters deliver trailers to docks, where they are processed

(stage two) and finally delivered back to the parking lot (stage three). A peculiarity is that stages one and three share resources, i.e., the spotter fleet. Mixed integer models for the resulting problem are provided in Berghman et al. (2014) and heuristic solutions for a real-world case at a Toyota warehouse in Belgium are presented in Berghman and Leus (2015).

Clearly, truck and spotter scheduling are heavily interdependent, so that integrating both planning tasks into a holistic problem seems a good choice for many settings. However, the resulting problem is quite bulky and difficult to solve. Therefore, it may be convenient to decompose it if a single stage of the three-stage process is the main bottleneck. This is, for instance, the case at the aforementioned yard of a large German postal service provider we visited. Over the years especially the increasing volumes of e-commerce have led to a much higher workload to be processed at the terminal. While the spotter fleet can conveniently be adapted to changing capacity situations, the terminal cannot be expanded that easily, so that (especially during peak hours) the dock doors are the unique bottleneck resource. In such a setting, truck schedules can be derived independently of the spotters, such that the docks are efficiently utilized. Once such a truck schedule is given, the aim of the spotter schedule reduces to realizing the given trailer transports in a timely manner such that the docks never run idle.

Instead of tightly coupling trailer processing and spotter scheduling, which may lead to a propagation of delays whenever unforeseen events occur during plan execution, the focus should rather be on robustness. Although the spotter fleet is sufficiently large per se, from time to time unforeseen delays occur in our terminal. For instance, unloading a trailer may take a little bit more time than expected or some trucks may arrive late at the terminal. These unforeseen events that regularly occur in any terminal lead to an idle time of the actual bottleneck resource, i.e., the dock doors. Therefore, we aim to determine, for the given truck schedules, spotter schedules that are robust in the face of unforeseen disturbances.

Although robust optimization is a wide and ever-expanding field of research and there exist quite a few (partially conflicting) definitions, a general and widely accepted description goes like this (Briskorn et al. 2011): Robust optimization is the task of finding a feasible solution to an optimization problem that is not necessarily optimal but remains feasible (solution-robustness) and has a good performance (quality-robustness) if parameter values of the problem change. Surveys on the vast amount of literature on robust optimization are, for instance, provided by Ben-Tal and Nemirovski (2002), Ben-Tal and Nemirovski (2008) and Kouvelis and Yu (1997). Robust scheduling in particular is surveyed by Aytug et al. (2005). Among the manifold approaches to reach robust schedules, a convenient way, which often leads to still comparatively compact problems, is the insertion of time buffers. A buffer between two jobs executed by the same resource protects the start time of the latter job against delays of the completion time of the former one. Time buffers have, for instance, been considered in project scheduling (e.g., Herroelen and Leus 2004; Van de Vonder et al. 2005) and machine scheduling (e.g., Leus and Herroelen 2007; Briskorn et al. 2011). We, however, have fixed execution intervals for our jobs (i.e., trailer movements), so that time buffers are only influenced by their assignment to resources (i.e., spotters). Fixed processing intervals are the topic of interval

scheduling (see Kolen et al. 2007 for a survey), where to the best of our knowledge robustness and time buffers have not been considered yet.

The resulting robust interval scheduling problems are general problems, which are not bound to spotter scheduling, but can be applied whenever predefined (transport) jobs with given processing intervals have to be executed in a timely manner by a given set of machines (or vehicles). However, in the following, we stick to the case of spotter scheduling because in this area of application the need to derive robust plans was brought to our attention.

1.2 Contribution and paper structure

This paper treats robust spotter scheduling by inserting time buffers between job pairs successively executed by the same spotter. Specifically, we assign transport jobs with given processing intervals to a fixed fleet of spotters under two robustness objectives: (i) maximize the minimum time buffer and (ii) maximize the sum of time buffers. The resulting two optimization problems are defined in Sect. 2. Polynomial-time algorithms for solving both problems to optimality are provided in Sect. 3. Our computational study (Sect. 4) simulates different disturbances during plan execution and evaluates the ability of our two robustness approaches to avoid turmoil in the yard. Our results show that robust schedules, especially if optimized according to objective (i), can successfully avoid that delays of single trailer movements or delayed external trucks lead to a propagation effect causing further delays for subsequent yard processes. In this way, the average waiting time per truck and the average idle time per dock door reduces by more than 1 and 5 h, respectively, compared to non-robust solutions. Finally, Sect. 5 concludes the paper.

2 Problem description

Consider a set $J = \{1, \ldots, n\}$ of jobs, each representing a fixed transport request defined by its completion time C_j, a processing time p_j, and a weight w_j. Jobs represent transport requests either from a parking position to a dock or vice versa. The input parameters of our problem can directly be derived from the output of truck scheduling problem, which we assume to be pre-defined. The truck schedule for each truck determines when and where it has to be (un-)loaded, i.e., it assigns each trailer to a specific dock door and fixed processing time interval. As we assume that the parking positions for all trailers are also given, we can preprocess the travel time it takes a spotter to transport the trailer between the given dock and the respective parking position (i.e., p_j). A transport job of a trailer toward a dock consists of coupling a trailer to a spotter at the initial parking position and moving it toward the dock, where it is uncoupled. A retrieval of the same trailer after its processing reverts this flow: a spotter is coupled to the trailer at its dock, moves it toward the parking position, and uncouples it. It is thus easy to derive the processing time p_j and the completion time C_j of each job j so that it is executed in a timely manner in order to meet the fixed predefined (un-)loading time intervals. The weights w_j

denote the importance of a job being processed on time. E.g., an important truck that is fully loaded with high-priority freight may receive a greater weight than non-urgent deliveries. Furthermore, we have sequence-dependent setup times $\delta_{jj'}$, which represent the deadheading time it takes a spotter to move from the target position of job j to the start position of job j'. Note that we assume that the first job of each spotter can be processed without any driving time. As a matter of convenience, we assume jobs being numbered according to non-decreasing start times, i.e., $C_1 - p_1 \leq C_2 - p_2 \leq \cdots \leq C_n - p_n$.

Given job set J and a fixed fleet of spotters $S = \{1, \ldots, m\}$, a schedule is defined by a partition $\{R_1, \ldots, R_m\}$ of J and a permutation ω_s of R_s, $\forall s \in S$, specifying the order in which the jobs assigned to spotter s are executed. Let $\omega_s(l) \in R_s$ denote the l-th job of spotter s, i.e., $\omega_s = \langle \omega_s(1), \ldots, \omega_s(l), \ldots, \omega_s(|R_s|)]$. Moreover, let $\eta(j) \in S$ be the spotter job $j \in J$ is assigned to, and let $\theta(j) \in \{1, \ldots, |R_{\eta(j)}|\}$ be the sequence position of job j. Then, we define the buffer time $b(j)$ as the amount of idle time between job j and its predecessor $\omega_{\eta(j)}(\theta(j) - 1)$. If j is the first job executed by a spotter (i.e., $\theta(j) = 1$), it does not have a predecessor, therefore we set the buffer time to the start time of that job. Hence, we have

$$b(j) = \begin{cases} C_j - p_j, & \text{if } \theta(j) = 1 \\ C_j - p_j - \delta_{\omega_{\eta(j)}(\theta(j)-1),j} - C_{\omega_{\eta(j)}(\theta(j)-1)}, & \text{else} \end{cases},$$

$\forall j \in J$.

For simplicity's sake, we neglect the spotters' initial driving times ahead of their first job. Thus, we assume that each spotter is initially directly available at its first job at the beginning of the planning horizon. Note that if for each spotter $s \in S$ a specific initial state (i.e., an earliest availability time e_s and an initial driving time δ^s_{0j} toward each job $j \in J$) has to be considered, the buffer time of each initial job j (i.e., if $\theta(j) = 1$) can be computed as $b(j, s) = C_j - p_j - \delta^s_{0j} - e_s$ for each spotter $s \in S$. Such a setup would allow multiple schedule updates throughout the day, e.g., when planning on rolling planning horizons. In our baseline model, however, we do not explicitly take the spotters' initial states into account which remains a valid task for future research.

We say that a schedule is feasible if it satisfies the following conditions.

1. Each spotter is assigned at least one job, i.e., $R_s \neq \emptyset$, $\forall s \in S$. Note that it can obviously not be optimal with regard to robustness to have unused spotters. Also note that this presupposes that $n \geq m$, which can always be imposed by "culling" excess spotters from the instance if necessary.
2. The buffer time between any two jobs must not be negative (no spotter can perform two jobs at the same time), i.e., $b(j) \geq 0$, $\forall j \in J$.

The main idea of a buffer time $b(j)$ between job j and its predecessor $j' = \omega_{\eta(j)}(\theta(j) - 1)$ is to protect the start time of j. If an unforeseen prolongation of unloading some truck leads to a delay of job j', then the start time of its successor

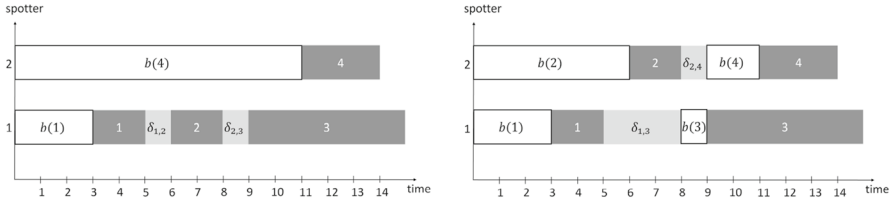
j is not affected if the delay is not greater than $b(j)$. Otherwise, j cannot start as planned but is delayed less than it would be without a buffer. This way, a diffusion effect, i.e., the propagation of a delay at one dock to other docks via delayed spotters, is avoided or at least mitigated. Specifically, we consider the time buffers in two different robustness objectives.

- Objective max-$\sum$ maximizes the total weighted buffer time, i.e., $F^{\text{sum}} = \sum_{j \in J} w_j \cdot b(j)$.
- While the former objective maximizes the overall buffer time, it does not preclude individual buffers from being small (or even zero). To afford each job some level of protection, objective max–min maximizes the minimum buffer size, i.e., $F^{\min} = \min\{b(j)/w_j \mid j \in J; \theta(j) > 1\}$. Note that, for this objective, only the buffer times between jobs (as opposed to the start time of the first job) are considered because the buffer before the first job only depends on its given start time and thus should not influence the objective.

Example: Consider an example problem with $n = 4$ transport jobs to be processed by $m = 2$ spotters. The processing (p_j) and completion (C_j) times as well as the weights (w_j) of each job are in Table 1a. The driving time from the end point of some job j to the starting point of any other job j', i.e., $\delta_{jj'}$, can be found in Table 1b. For this problem, the optimal solution with regard to objective max -$\sum$ is $\omega_1 = \langle 1, 2, 3]$ and $\omega_2 = \langle 4]$, that is, one spotter performs job 1, then job 2 and job 3, while the other spotter handles job 4. This leads to buffer times of $b_1 = 3$, $b_2 = 0$, $b_3 = 0$, and $b_4 = 11$, and hence to a total objective value of $F^{\text{sum}} = 1 \cdot 3 + 2 \cdot 0 + 1 \cdot 0 + 3 \cdot 11 = 36$. The solution is depicted as a Gantt chart in Fig. 2a. Using the max–min objective, the optimal solution becomes $\omega_1 = \langle 1, 3]$ and $\omega_2 = \langle 2, 4]$, implying buffer times of $b_1 = 3$, $b_2 = 6$, $b_3 = 1$, and $b_4 = 2$, and hence, an objective value of $F^{\min} = \min\{1/1, 2/3\} = 2/3$. Figure 2b shows the corresponding Gantt chart.

Table 1 Example problem: input parameters

j	1	2	3	4
(a) Jobs' characteristics				
p_j	2	2	6	3
C_j	5	8	15	14
w_j	1	2	1	3
$\delta_{jj'}$	1	2	3	4
(b) Driving times				
1	–	1	3	2
2	1	–	1	1
3	3	1	–	4
4	2	1	3	–



(a) Gantt chart for max-$\sum$ example. **(b)** Gantt chart for max-min example.

Fig. 2 Example problem: optimal solutions

3 Algorithms

In this section, we will present polynomial-time exact algorithms for the robust spotter scheduling problem (RSSP), both under the max-$\sum$ as well as the max –min objective.

3.1 Objective max–$\sum$

Maximizing the total buffer time, RSSP can be reduced to maximum weight perfect matching in a bipartite graph.

Let $G(U \cup V, E, c)$ be a bipartite graph, where U and V are two disjoint vertex sets, E is a set of edges, with each edge connecting one vertex in U to one vertex in V, and $c : E \to \mathbb{R}_+$ is a weight function. $U = \{u_1, \ldots, u_n, \lceil_1, \ldots, \lceil_m\}$ and $V = \{v_1, \ldots, v_n, \rfloor_1, \ldots, \rfloor_m\}$ each contain one vertex for each of the n elements in job set J and m additional "dummy" vertices. We say that vertices v_j and u_j correspond to job j, $\forall j = 1, \ldots, n$, and vertices $\lceil_l$ and $\rfloor_l$ correspond to spotter l (note that spotters are homogeneous and the indices for the latter are therefore interchangeable).

Each "dummy" vertex $\lceil_l$ in set U is connected to each job vertex v_j in set V by an edge $(\lceil_l, v_j) \in E$, $\forall l = 1, \ldots, m; j = 1, \ldots, n$. Edge $(\lceil_l, v_j)$ indicates that job j is the first job to be performed by spotter l. Consequently, the weight of that edge is $c(\lceil_l, v_j) = w_j \cdot (C_j - p_j)$.

Each job vertex u_j in set U is connected to a job vertex $v_{j'}$ in set V by an edge $(u_j, v_{j'}) \in E$ if and only if job j' can be performed after job j by the same spotter, i.e., if $C_{j'} - p_{j'} - \delta_{j,j'} - C_j \geq 0$. Such an edge stands for one spotter performing job j' immediately after job j. In that case, the edge weight is $c(u_j, v_{j'}) = w_{j'} \cdot (C_{j'} - p_{j'} - \delta_{j,j'} - C_j)$, i.e., the weighted time buffer between jobs j and j', given that they are processed in direct succession by the same spotter.

Finally, each job vertex u_j in set U is connected to each "dummy" vertex $\rfloor_l$ in set V by an edge $(u_j, \rfloor_l) \in E$, $\forall j = 1, \ldots, n; l = 1, \ldots, m$, indicating that job

j is the last job performed by a spotter. The corresponding edge weight is then $c(u_j, \rceil_l) = 0$.

This definition of G allows stating the following proposition.

Lemma 3.1 *A maximum weight perfect matching in graph G corresponds to a feasible RSSP solution which is optimal with regard to the* max-$\sum$ *objective and vice versa.*

Proof A feasible RSSP solution can easily be constructed from a perfect matching in G: If some edge $(\lfloor_l, v_j)$ is part of the matching, it means that spotter l starts off with job j. If then there is an edge $(u_j, v_{j'})$ in the matching, that means that job j' is the direct successor of j. Finally, if there is an edge $(u_j, \rceil_{l'})$, it means that job j is the last job of spotter l. Note that l' need not necessarily be equal to l. Also, note that every job must be performed by exactly one spotter, and each spotter must perform at least one job, else the matching would not be perfect. Finally, seeing that edge weights correspond to weighted buffer times, it is clear that a maximum weight matching implies optimality with regard to F^{sum}.

By the same logic, this transformation can also be applied in reverse to create a maximum weight perfect matching in G from a feasible RSSP solution which is optimal with regard to F^{sum}. □

Example (cont.): The maximum weight matching for our example is graphically depicted in Fig. 3, corresponding to the same solution presented in Fig. 2a.

As the number of (potentially useful) spotters m is bounded by the number of jobs n, the total number of vertices in G is bounded by $O(n)$ and the total number of edges by $O(n^2)$. Consequently, using, for example, the improved Hungarian algorithm [e.g., Burkhard et al. 2009, Ch. 4], we get the following proposition.

Proposition 3.1 *RSSP with* max-$\sum$ *objective can be solved in* $O(n^3)$ *time.*

3.2 Objective max–min

We split solving RSSP with $F^{\min}$ objective into two parts. First, we will discuss how to find a feasible solution to RSSP (if one exists) for some given minimum time buffer $\overline{F}^{\min}$, i.e., a feasible solution where $b(j)/w_j \geq \overline{F}^{\min}, \forall j \in J; \theta(j) > 1$.

This problem can again be reduced to finding a perfect matching in a bipartite graph. Let $\mathcal{G}(U \cup V, \mathcal{E})$ be a bipartite graph, where U and V are two disjoint vertex sets and $\mathcal{E}$ is a set of edges, each connecting a node in U to a node in V. Sets U and V are set up exactly as described in Sect. 3.1.

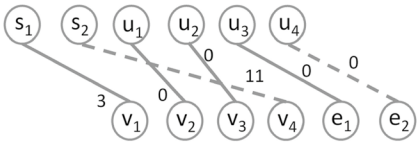


Fig. 3 Optimal maximum weight matching in the example

Set $\mathcal{E}$ contains edges $(\lceil_l, v_j)$ as well as $(u_j, \rceil_l)$, $\forall l = 1, \dots, m; j = 1, \dots, n$, indicating that job j is the first or last job executed by a spotter, respectively. Moreover, there is an edge $(u_j, v_{j'})$ if and only if $(C_{j'} - p_{j'} - \delta_{j,j'} - C_j)/w_{j'} \geq \overline{F}^{\min}$, meaning that job j' can only be performed right after job j by the same spotter if the weighted time buffer between these two jobs is not below the given minimum $\overline{F}^{\min}$.

Lemma 3.2 *There is a perfect matching in $\mathcal{G}$ if and only if there is a feasible RSSP solution where $b(j)/w_j \geq \overline{F}^{\min}$, $\forall j \in J; \theta(j) > 1$.*

Proof If there is a perfect matching in $\mathcal{G}$, it can be transformed to a feasible RSSP solution exactly as explained in the proof of Lemma 3.1. In such a solution $b(j)/w_j \geq \overline{F}^{\min}$, $\forall j \in J; \theta(j) > 1$, must hold because of the way set $\mathcal{E}$ is constructed.

Conversely, if there is a feasible solution where $b(j)/w_j \geq \overline{F}^{\min}$, $\forall j \in J; \theta(j) > 1$, holds, there must also be a perfect matching in the corresponding graph $\mathcal{G}$. In such an RSSP solution, every job must clearly be the first or the last job of exactly one spotter, and/or it must be the successor of exactly one other job, such that the weighted time buffer between those jobs does not fall below $\overline{F}^{\min}$. In either case, the corresponding edge is part of $\mathcal{E}$. Since this holds for every job in the RSSP solution, $\mathcal{G}$ must permit a perfect matching. □

Solving the matching problem as it is outlined above allows constructing a feasible solution for a given objective value of $\overline{F}^{\min}$. If the optimal objective value F^{min*} was known, it would, therefore, be possible to generate a corresponding optimal solution. To find the optimal objective value, we propose the scheme outlined in Algorithm 1. This procedure is essentially a binary search scheme: The optimal max–min objective value must be one of the possible weighted time buffers between jobs, i.e., $F^{min*} = (C_{j'} - p_{j'} - \delta_{j,j'} - C_j)/w_{j'}$, for some $j, j' \in J; j' > j$. In Algorithm 1, the set of these potentially optimal $\overline{F}^{\min}$ values is denoted as B. Sorting these values in ascending order (list $\lfloor$ in Algorithm 1), finding the greatest feasible and hence optimal $\overline{F}^{\min}$ can be implemented as a binary search on $\lfloor$. If a given $\overline{F}^{\min}$ value does not permit a feasible solution, the $\overline{F}^{\min}$ value must be too ambitious and is lowered in the next iteration. On the other hand, if a feasible solution can be constructed, then the current $\overline{F}^{\min}$ may be too low and it is increased in the next iteration.

Input: instance of RSSP
$B := \emptyset$; // set of different buffer values
for $j = 1$ **to** $n - 1$ **do**
  for $j' = j + 1$ **to** n **do**
    if $C_{j'} - p_{j'} - \delta_{j,j'} - C_j \geq 0$ **then**
      $B := B \cup \{(C_{j'} - p_{j'} - \delta_{j,j'} - C_j)/w_{j'}\}$;
$\mathcal{b} :=$ sorted list containing the elements in B in ascending order;
$q^{min} := 1$;
$q^{max} := |B|$;
repeat
  $q := \lfloor (q^{min} + q^{max})/2 \rfloor$;
  $\overline{F}^{min} :=$ the q-th element from $\mathcal{b}$;
  try to find a perfect matching using $\overline{F}^{min}$ as per Lemma 3.2;
  if *perfect matching exists* **then**
    $q^{min} := q + 1$;
    $\Omega :=$ RSSP solution derived from perfect matching;
  else
    $q^{max} := q - 1$;
until $q^{min} > q^{max}$;
Output: RSSP solution Ω that is optimal with regard to the max-min objective

Algorithm 1: Solving RSSP with max-min objective.

Example (cont.): The potentially optimal values, sorted in ascending order, for $\overline{F}^{\min}$ are $\lfloor = \langle 0, 2/3, 1, 4/3]$. The greatest feasible value from $\lfloor$ is 2/3. The perfect matching for $\overline{F}^{\min} = 2/3$ is depicted in Fig. 4, corresponding to the same solution presented in Fig. 2b.

This leads us to the following proposition.

Proposition 3.2 *RSSP with* max–min *objective can be solved in* $O(n^3 \log n)$ *time.*

Proof The set of possible $\overline{F}^{\min}$ values, B, can contain at most $O(n^2)$ distinct values. Since binary search halves the search space in each iteration, it can sift through these values in $O(\log n^2)$ iterations, and in each iteration, a perfect matching problem in a bipartite graph has to be solved, which can be done in $O(n^3)$ time. Consequently, the total asymptotic runtime comes to $O(n^3 \log n)$.

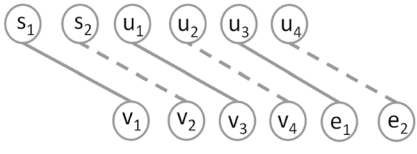


Fig. 4 Perfect matching for $\overline{F}^{\min} = 2/3$

4 Computational study

The objective of our computational study is to answer the following research questions. First, what is the computational performance of our proposed solution methods with regard to both objectives and different yard sizes (Sect. 4.2)? Second, do our surrogate objectives indeed promote robustness to operational disturbances in a yard terminal? Since no established testbed for RSSP instances exists, we describe how our test instances are generated in Sect. 4.1. To address the second research question, we apply a terminal simulation, whose setup is described in Sect. 4.3. In the simulation study, we investigate the robustness of RSSP solutions in case of unexpected disturbances (Sect. 4.4). Specifically, we observe the effects of internal delays, i.e., unexpectedly prolonged (un-)loading times at docks, and external delays, i.e., delayed arrivals of trailers at the terminal.

4.1 Instance generation

Our instance generation scheme is geared to the hub terminal of the aforementioned postal service provider. Specifically, we first generate realistic truck schedules, and then, based on those truck schedules, the corresponding RSSP instances are derived.

RSSP is an operational planning problem, which has to be solved once truck schedules (i.e., the where and when of trailer processing at the dock doors) are known. The typical planning horizon for the truck scheduling problem is one day (Boysen et al. 2017). Thus, it is typically pointless to plan spotter schedules more than one day in advance. Moreover, to reduce forecasting errors of input parameters (e.g., truck arrival times, amount, and type of trailer loads, and their (un-)loading durations), it may be advisable to schedule spotters for an even shorter planning horizon (e.g., one or even half a work-shift). Given the operational character of RSSP, we aim to generate precise spotter schedules. Therefore, to avoid rounding errors, a time unit in an RSSP instance corresponds to one second of realtime.

Our trailer yard consists of a terminal and a parking lot. The terminal contains a set D of dock doors where the goods are to be unloaded from incoming trailers and/or loaded into outgoing trailers. The parking lot is divided into L parking lines, each of which is located parallel to the docking wall and contains $|D|$ parking positions. Thus, a set Π of parking positions in the terminal yard consists of $|\Pi| = |D| \cdot L$ positions. Our instance generator receives the number of dock doors $|D|$ and the number of parking lines L as input parameters that define the trailer yard size according to the dimensions of our real-world terminal.

We assume that there are two-way roads in the whole yard so that spotters can move in both directions everywhere. This way, we can easily derive the distance $d_{\alpha,\beta}$ between any pair of positions α and β in the trailer yard ($\alpha, \beta \in \{D \cup \Pi\}$). The schematic layout of a trailer yard with $|D| = 5$ dock doors and $L = 3$ parking lines is depicted in Fig. 5. Note that a detailed description of how the distances in the yard are calculated is given in the "Appendix".

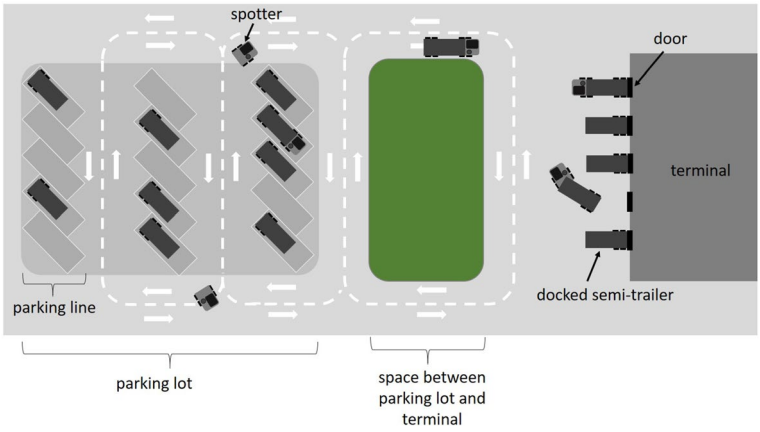


Fig. 5 Schematic layout of the trailer yard for instance generation

In line with the planning parameters applied at our real-world case, we assume that a spotter without attached semi-trailer moves with a constant speed of $v^f = 5$ m/s (i.e., 18 km/h) while the speed of a spotter coupled with a semi-trailer is assumed to be $v^b = 3$ m/s (i.e., 10.8 km/h). Furthermore, we assume that semi-trailer coupling and uncoupling (including parking and pulling out) always last $\Delta^c = 30$ and $\Delta^u = 20$ s, respectively.

As a first step, to generate a feasible truck schedule for a given set of trucks I, for each truck $i \in I$ we randomly determine its arrival time a_i, parking position π_i in the yard, and handling time h_i at the dock door (i.e., duration of the (un-) loading of the trailer). For the sake of simplicity, we assume that the number of trucks $|I|$ in the planning horizon is equal to the total number of parking positions in the parking lot ($|I| = |\Pi|$). This way, each truck $i \in I$ is associated with exactly one parking space and vice versa. Note that most large trailer yards are located in rural areas where land is not that costly. Thus, this simplifying assumption should seldom be a shortcoming. We assign each trailer i to a parking position π_i randomly while taking into account that to each parking position π_i exactly one trailer is assigned ($\pi_i \neq \pi_{i'} | \forall i, i' \in I; i \neq i'$).

It is well-known that the truck scheduling problem can be modeled as a parallel machine scheduling problem (Tadumadze et al. 2019, 2020). Thus, we generate arrival times and (un-)loading times of trucks similarly to the instance generation scheme proposed by Hall and Posner (2001). Specifically, handling time h_i for each trailer $i \in I$ is drawn from a normal distribution with an expected handling time of 30 min and variance of 6 min (i.e., $\mu = 1800, \sigma^2 = 6$). Furthermore, to generate arrival times a_i for each truck $i \in I$, the inter-arrival times of two successive trucks are randomly drawn from an exponential distribution with mean $\lambda = \frac{\mu}{|D|}$ (starting at $a_1 = 1800$). Note that all input parameters are assumed to have deterministic values.

To derive a truck schedule, we first sort trailers according to their arrival times a_i and then assign each trailer $i \in I$ to the next free dock door $g_i \in D$ and (un-)loading

time window "first come first serve." As a result, for each truck $i \in I$ start (s_i) and end ($e_i = s_i + h_i$) of (un-)loading is determined.

Finally, for each trailer $i \in I$, the relative importance r_i of its truckload is drawn randomly with a uniform distribution from the interval [1, 5].

Each trailer i is associated with exactly two jobs for spotters j and j':

- j: transport of trailer i from parking position π_i to dock door g_i with processing time $p_j = \Delta^c + \frac{d_{\pi_i,g_i}}{v^b} + \Delta^u$ (i.e., transport time of the trailer from the parking position to the dock door) and completion time $C_j = s_i$ (i.e., the trailer transport job is completed when the trailer has been set down at the door and its (un-)loading starts),
- j': return the trailer from g_i to π_i with $p'_j = \Delta^c + \frac{d_{g_i,\pi_i}}{v^b} + \Delta^u$ and $C'_j = e_i + p'_j$.

Setup times $\delta_{jj'}$ for each pair of jobs $j, j' \in J$ are determined by dividing $d_{\beta_j,\alpha_{j'}}$ (the distance between the end position β_j of job j and the start position $\alpha_{j'}$ of job j') by constant speed v^f of an empty spotter, i.e., $\delta_{jj'} = \frac{d_{\beta_j,\alpha_{j'}}}{v^f}$. The weight of each job $j \in J$ is derived as $w_j = r_i \cdot rnd[0.5, 2]$ where rnd denotes a uniformly distributed random number from the interval in the argument. This way, the weight of the job on the one hand depends on the relative importance of the corresponding trailer and, on the other hand, it is randomly re-weighted in respect of some other job-specific issues (e.g., urgency at the corresponding dock door, or a pressing truck departure time).

We have implemented our algorithms, including the shortest augmenting path algorithm by Jonker and Volgenant (1987) for finding a (maximum weighted) perfect matching in a bipartite graph, as well as the simulation model in C# 6.0. All tests have been executed on an x64 PC with an Intel Core i7-8700K 3.70 GHz CPU and 64 GB RAM. The generated RSSP instances are available and can be downloaded using the following DOI: 10.5281/zenodo.3925356.

4.2 Computational performance

In this section, we investigate and compare the computational performance of the proposed algorithms (i.e., max-$\sum$ and max–min). To do so, we generate RSSP instances with our instance generator. Specifically, we assume three differently sized terminals consisting of $|D| = 20$, $|D| = 50$, and $|D| = 200$ doors (in the rest of the text the three different terminal sizes are dubbed S, M, and L). For each terminal size, we presuppose four different numbers $L \in \{2, 3, 4, 5\}$ of parking lines, which together with $|D|$ determine the size of the parking yard $|\Pi|$, the number of trucks $|I|$, and the number of transport jobs $n = 2 \cdot |I|$ in the planning horizon. Furthermore, for each terminal size, we assume three different spotter fleet sizes m. The applied input parameters are summarized in Table 2. For each parameter combination, our instance generator generates 10 RSSP instances, so that in total there are 360 RSSP instances for the computational performance test.

Table 3 summarizes the aggregated computational results of the algorithmic performance test. Each row in Table 3 contains the results averaged over the 10 problem instances of the same size. For each algorithm (i.e., max-$\sum$ and max–min), we

Table 2 Parameters for instance generation of computational performance test

Parameter	Terminal size		
	S	M	L
$\|D\|$	20	50	200
L	(2, 3, 4, 5)	(2, 3, 4, 5)	(2, 3, 4, 5)
$\|I\|$	(40, 60, 80, 100)	(100, 150, 200, 250)	(400, 600, 800, 1000)
n	(80, 120, 160, 200)	(200, 300, 400, 500)	(800, 1200, 1600, 2000)
m	(20, 30, 40)	(50, 75, 100)	(200, 300, 400)

report the average computational time that is required to obtain the optimal solution (column "CPU sec"). For comparison, we also evaluate both objective values $F^{\min}$ and F^{sum} (columns "$F^{\min}$" and "F^{sum}," respectively), even if the other objective function is pursued. Thus, we solve RSSP under the max-$\sum$ objective and evaluate the resulting optimal solution with the objective function of the max–min problem (and vice versa). Note that the $F^{\min}$ values are only optimal when the max–min problem is solved; analogous for F^{sum} and max-$\sum$.

Our experiment shows that both algorithms are well suited for solving RSSP to optimality in short computational time. Even in the most extreme cases, the computational times of the hardest problem instances for the large terminal yard (scheduling $n = 2,000$ jobs on $m = 300$ or $m = 400$ spotters) do not exceed 1 min when total weighted buffer time (i.e., objective max-$\sum$) is maximized and 8 min in case of objective max–min, respectively. For all problem instances, the computational times under the max-$\sum$ objective are always shorter compared to the max–min case. On the other hand, maximizing the total weighed buffer time apparently comes at the cost of low-weighted jobs receiving very short or even no buffer times. This can be seen in the $F^{\min}$ values of the solutions where the $\min - \sum$ objective is applied, which are always close to zero (column "$F^{\min}$" for max-$\sum$ objective). This may damage the robustness of solutions, because in case of unforeseen events during plan execution delays can propagate. We investigate this further in Sect. 4.4.

Unsurprisingly, the terminal size strongly affects the computational times of both algorithms. The average computational times of both algorithms for the three terminal sizes (i.e., S, M, and L) are depicted in Fig. 6. We further observe the impact of n (i.e., the number of transport jobs) and m (i.e., the number of spotters) on problem complexity. According to the experimental results, the spotter fleet size m has only a negligible impact on the problem complexity, while the number of transport jobs n seems to have a considerable influence on the computational time. Figure 7 depicts the average computational times for instances with varying numbers of jobs n for each terminal size. In all cases, with the increase in parameter n the average computational times of both algorithms increase superlinearly, although max-$\sum$ scales better than max–min.

It can be concluded from the performance test that our solution algorithms seem well suited for practical applications. They deliver optimal solutions quickly

Table 3 Results of computational performance test

Instance size		max–$\sum$			max–min		
m	n	CPU sec	F^{sum}	$F^{\min}$	CPU sec	F^{sum}	$F^{\min}$
Terminal size: S							
20	80	0.01	673,732.80	0.10	0.02	529,380.10	109.30
30	80	0.01	887,759.20	0.20	0.02	754,772.10	259.33
40	80	0.01	1,067,874.70	2.50	0.02	976,983.00	284.50
20	120	0.02	957,800.50	0.03	0.05	688,786.00	107.75
30	120	0.02	1,363,439.70	0.00	0.05	1,071,680.30	221.38
40	120	0.02	1,616,270.30	0.00	0.06	1,333,432.80	278.46
20	160	0.04	1,130,077.30	0.00	0.12	777,390.70	90.10
30	160	0.04	1,640,356.70	0.00	0.12	1,194,765.80	210.72
40	160	0.04	2,065,274.30	0.00	0.13	1,617,540.10	277.67
20	200	0.06	1,316,235.40	0.00	0.24	834,793.70	61.13
30	200	0.07	2,030,708.50	0.00	0.26	1,442,855.30	213.05
40	200	0.07	2,577,453.60	0.00	0.27	1,947,120.30	281.76
Mean (S)		*0.03*	*1,443,915.25*	*0.24*	*0.11*	*1,097,458.35*	*199.60*
Terminal size: M							
50	200	0.08	2,801,623.40	0.35	0.27	2,234,450.30	271.04
75	200	0.08	3,384,937.90	0.65	0.29	2,958,384.40	261.79
100	200	0.08	3,999,989.70	1.68	0.32	3,732,210.60	286.27
50	300	0.22	3,895,340.30	0.00	0.95	2,843,258.20	268.39
75	300	0.24	5,458,433.00	0.18	1.07	4,367,426.00	288.90
100	300	0.25	6,300,740.30	0.33	1.09	5,420,223.30	280.31
50	400	0.46	5,142,676.80	0.00	2.41	3,567,202.20	277.59
75	400	0.49	7,124,042.70	0.03	2.53	5,354,420.40	285.28
100	400	0.53	8,992,119.00	0.00	2.70	7,223,840.40	298.59
50	500	0.83	6,435,699.20	0.00	4.97	4,265,949.10	251.84
75	500	0.90	9,238,427.70	0.08	5.28	6,603,872.90	266.86
100	500	0.96	11,516,821.00	0.08	5.56	8,846,761.80	297.87
Mean (M)		*0.43*	*6,190,904.25*	*0.28*	*2.29*	*4,784,833.30*	*277.89*
Terminal size: L							
200	800	4.09	32,045,392.00	0.00	26.34	25,975,279.00	272.16
300	800	4.40	38,253,184.00	0.13	27.60	34,427,745.00	278.55
400	800	4.54	46,529,532.00	0.25	30.00	44,087,069.00	266.96
200	1200	11.97	53,432,412.00	0.00	93.21	39,669,969.00	274.18
300	1200	13.21	68,786,194.00	0.00	97.63	56,381,906.00	276.93
400	1200	14.17	79,158,212.00	0.00	100.31	69,536,510.00	265.67
200	1600	26.16	73,286,205.00	0.00	226.35	51,825,850.00	265.62
300	1600	29.02	102,827,090.00	0.00	241.58	79,509,343.00	286.44
400	1600	30.82	119,421,480.00	0.00	246.04	98,797,397.00	263.93
200	2000	48.74	96,623,679.00	0.00	456.86	65,889,463.00	264.14
300	2000	52.79	132,979,290.00	0.00	473.17	97,712,393.00	284.98
400	2000	56.47	158,843,290.00	0.00	481.95	125,592,390.00	269.38

Table 3 (continued)

Instance size		max–$\sum$			max–min		
m	n	CPU sec	F^{sum}	F^{min}	CPU sec	F^{sum}	F^{min}
Mean (L)		*24.70*	*83,515,496.67*	*0.03*	*208.42*	*65,783,776.17*	*272.41*

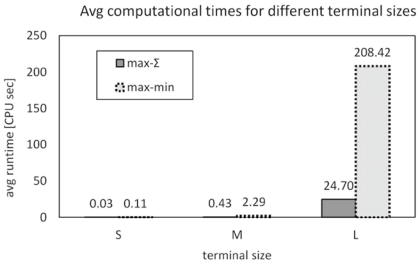


Fig. 6 Impact of terminal size on computational times

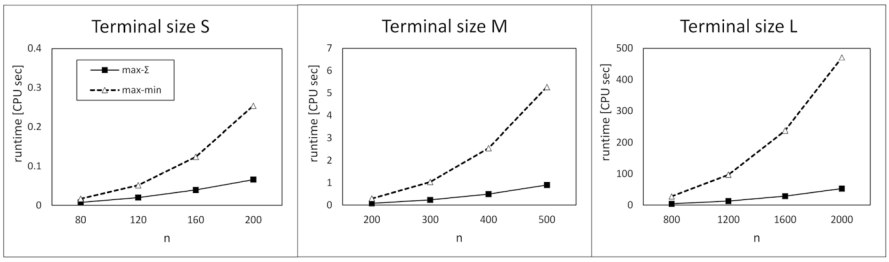


Fig. 7 Impact of n on computational times

even for very large yards, and the solution times are short enough to promptly return solutions whenever a replanning may be required during daily operations.

4.3 Setup of simulation study

In this section, we investigate whether RSSP solutions are actually robust against unforeseen delays that may occur during the execution of spotter schedules. To do so, we build a simulation model where some random events are simulated and the robustness of the solutions is evaluated. In the following, we describe the setup of our simulation model.

As a prototype terminal of our simulation model we select a medium-sized terminal with $|D| = 50$ doors, $|\Pi| = 300$ parking positions, and $|I| = 300$ trucks leading to $n = 600$ transport jobs. Furthermore, we vary parameter m (i.e., the number of spotters) between the following nine different values: $m \in \{20, 30, 40, 50, 60, 70, 80, 90, 100\}$. Again for each fleet size, we generate ten RSSP instances, so that for the simulation study we generated 90 RSSP instances in total.

We aim to observe the robustness of both objectives against unexpected disturbances. Therefore, for each generated instance we simulate some random delays. In particular, we consider the following two kinds of disturbances:

- *External disturbance* a^+: Some trucks may arrive later at the terminal yard than expected. The original arrival time a_i of delayed truck i is modified to $a'_i = a_i + a^+_i$, where a^+_i denotes the amount of delay of truck i.
- *Internal disturbance* h^+: The (un-)loading of some semitrailers may last longer than expected so that the handling time of such a trailer i at the dock is prolonged $h'_i = h_i + h^+_i$ with h^+_i defining the amount of delay.

For both kinds of disturbances our simulation model receives two predefined input parameters:

- the probability that a disturbance for a truck occurs: $p(a^+)$ and $p(h^+)$ as well as
- the maximal amount of delay [in min] that can occur: $\max(a^+)$ and $\max(h^+)$.

Given these parameters, for each truck i the simulation model first randomly decides whether or not it arrives with delay. If so, the amount of delay a^+_i is drawn randomly from a uniform distribution in the interval $[0, max(a^+_i)]$. Internal delays are determined analogously. To observe the effects of delays, we vary the input parameters in the following ranges $p(a^+) \in \{5, 10, 20\}$ [%] and $\max(a^+) \in \{10, 20, 30, 40, 50, 60\}$ [min] as well as $p(h^+) \in \{5, 10, 20\}$ [%] and $\max(h^+) \in \{10, 20, 30, 40, 50, 60\}$ [min]. The input parameters of our simulation study are summarized in Table 4.

After generating these random disturbances, an original RSSP instance P can be modified to P'. To do so, we reassign each truck $i \in I$ to the new (un-)loading time window according to the modified parameters a'_i and h'_i. Note, however, that we keep the door assignment of each truck as derived in the original truck schedule. This is because changing the door assignment on short notice triggers new problems inside the terminal. For instance, the goods dedicated to a specific outbound trailer are typically assembled (e.g., picked from a warehouse) over a longer period of time and placed in the outbound area directly in front of the

Table 4 Parameters for instance generation of simulation study

Parameter [unit]	Value(s)
$\|D\|$	50
L	6
$\|I\|$	300
n	600
m	(20, 30, 40, 50, 60, 70, 80, 90, 100)
$p(a^+)$ [%]	(5, 10, 20)
$\max(a^+)$ [min]	(10, 20, 30, 40, 50, 60)
$p(h^+)$ [%]	(5, 10, 20)
$\max(h^+)$ [min]	(10, 20, 30, 40, 50, 60)

respective dock door. Changing the dock door on short notice would require to move these goods to the new dock, which produces double handling and increases the probability of misplaced shipments.

Once our instance generator has produced an RSSP instance P, we solve it with regard to both objectives and store the corresponding optimal spotter schedules $\Omega(P)^*_{\text{sum}}$ and $\Omega(P)^*_{\text{min}}$. Moreover, to benchmark our robust spotter schedules against non-robust schedules, we additionally solve P to feasibility (by finding a perfect matching in the corresponding initial bipartite graph without considering any objectives) and store the resulting solution $\Omega(P)_{\text{feas}}$ too. Subsequently, our (simulated) disturbances occur, which leads to the modified RSSP instance P'. We evaluate the modified instance P' with the three fixed schedules (i.e., $\Omega(P)^*_{\text{sum}}$, $\Omega(P)^*_{\text{min}}$, and $\Omega(P)_{\text{feas}}$) and observe whether the new job completion times C'_j of P' are violated. Specifically, each spotter $s \in S$ executes the transport jobs in the given sequence ω_s according to the fixed spotter schedule ($\Omega(P)^*_{\text{sum}}$, $\Omega(P)^*_{\text{min}}$, or $\Omega(P)_{\text{feas}}$), which represent the schedules derived prior to the occurrence of disturbances. Given these fixed spotter schedules originally determined for P, we can derive whether our two robustness measures protect us against the disturbances of P'.

To compare the robustness of the schedules, we compute the lateness $l_j = \max\{0; \frac{C_j - t_j}{60}\}$ of each job $j \in J$ [measured in min], where t_j denotes the actual execution time of job j (i.e., the execution time of job j when P' is evaluated according to original spotter schedule $\Omega(P)^*_{\text{sum}}$, $\Omega(P)^*_{\text{min}}$, or $\Omega(P)_{\text{feas}}$). Then, based on the lateness of the job, we derive the following three performance measures, which evaluate the solution quality (i.e., robustness):

- *Average weighted lateness*: We define the average weighted lateness AWL of all jobs as the weighted sum of every jobs' lateness divided by the total number of jobs n: $\text{AWL} = \frac{\sum_{j \in J} w_j \cdot l_j}{n}$.
- *Truck-related lateness*: To reduce the delay of goods, terminal yards aim to provide (external) trucks with their (loaded or empty) semitrailers in a timely manner just as agreed. Thus, our external performance measure, average truck-related lateness TRL, calculates the total lateness of those jobs that are associated with bringing trailers from the terminal back to their parking position divided by the number of trucks $|I|$.
- *Dock-related lateness*: While the previous performance measure is geared to the external stakeholders of the yard, our third performance measure, average dock-related lateness DRL, is primarily relevant for the internal processes within the terminal. Specifically, DRL measures the total lateness of those transport requests whose target position is a dock door divided by the number of dock doors $|D|$. Such lateness leads to unforeseen waiting times at the dock doors and delays the processing of incoming goods inside the terminal.

The lower these three lateness measures for the schedules, the larger their protection level against unforeseen disturbances, which is evaluated in the following section.

4.4 On the impact of robust spotter scheduling

In this section, we benchmark the robustness of our two objectives max-$\sum$ and max–min if unforeseen disturbances occur during the execution of a spotter schedule. To do so, we solve the 90 generated RSSP instances to feasibility as well as with regard to both objectives and store the solutions. Then, we simulate disturbances, evaluate the modified data with the original spotter schedules derived for the initial (undisturbed) problem instance, and calculate our three lateness measures.

First, we observe robustness in case of only *external disturbances*, i.e., when some trucks arrive later than expected. For this, we simulate each problem instance in 18 different scenarios, where the values of external disturbance parameters, $p(h^+)$ and $\max(h^+)$ are varied in the ranges given in Table 4. Recall that the higher the former (latter), the more often external disturbances occur (the larger the delay of a late truck). Afterward, we address exclusively *internal disturbances*, i.e., some semitrailers require a longer processing time than expected. Here, we vary the probability $p(h^+)$ of the prolongation of a trailer's handling time at a dock and the maximal duration $\max(h^+)$ of the delay. Finally, we investigate the most realistic scenario when both kinds of disturbances—i.e., internal and external delays—occur simultaneously. Specifically, we simulate both kinds of delays at the same time by varying all parameters $p(a^+) = p(h^+)$, and $\max(a^+) = \max(h^+)$ in the ranges given in Table 4. As a result, each of the generated 90 instances is simulated in 54 different scenarios, each of which is solved and evaluated three times (to feasibility and with regard to both objectives).

Aggregated results of our simulation study are visualized in Figs. 8, 9 and 10. Specifically, Figs. 8 and 9 display the results for exclusively external and internal

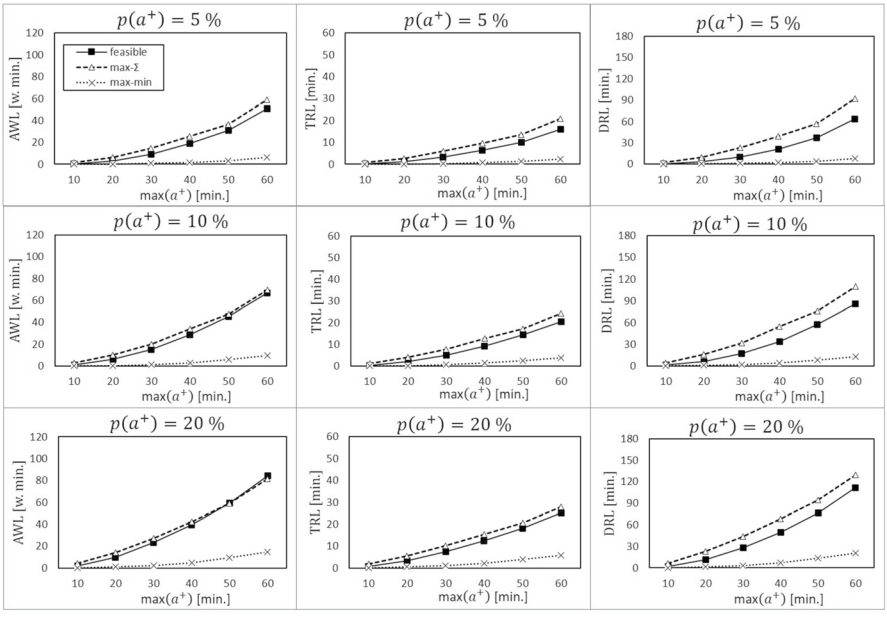


Fig. 8 Comparison of solution robustness in case of external delays

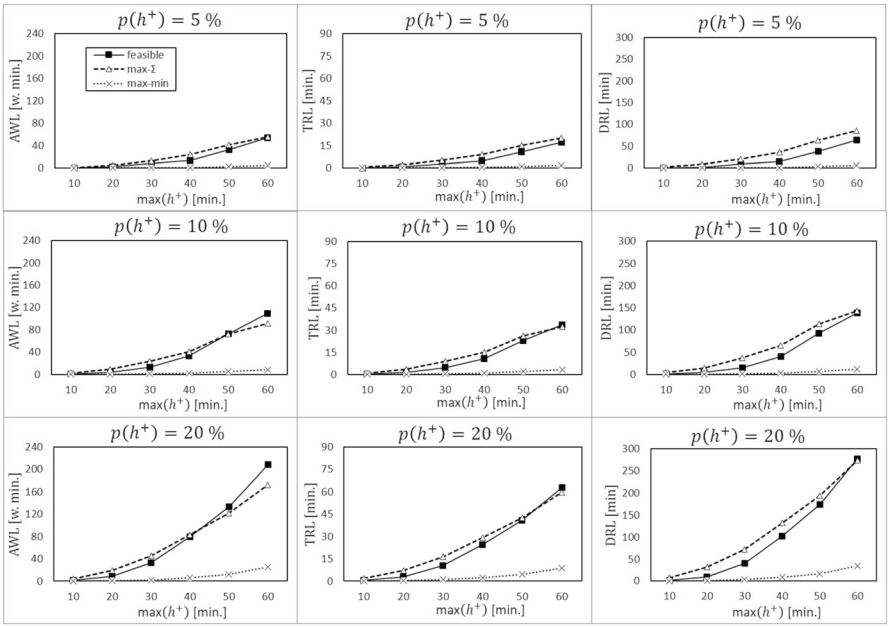


Fig. 9 Comparison of solution robustness in case of internal delays

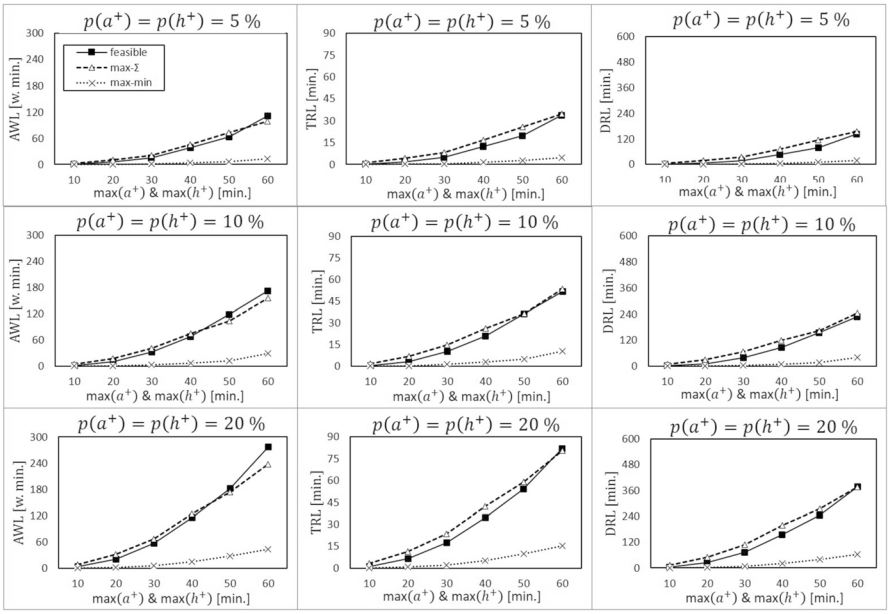


Fig. 10 Comparison of solution robustness in case of internal and external delays

disturbances, respectively, while Fig. 10 shows the results when both kinds of disturbances occur simultaneously.

Compared with non-robust solutions, applying the right robustness measure, i.e, the max–min objective, allows to protect against disturbances, and the lateness can be almost completely avoided (see slightly rising slopes of all three performance measures using the max–min objective). The optimal schedules with regard to our max-$\sum$ objective do not deliver results as good as those optimized using the max–min objective. This is due to objective max-$\sum$ generally favoring solutions where high-priority jobs (i.e., those with large weight w_j) receive generous buffers, while some or many other transport jobs have (almost) no buffer. Since any delay of a single job—even if it is a low-priority job—can lead to cascades of further delays of successive jobs, it is helpful to protect all jobs with buffer time, which is what the max–min objective promotes.

In general, the rising curves in all figures indicate that the robustness of initial schedules gets worse when the expected delay duration increases. Furthermore, the steeper slopes of the curves in scenarios with higher delay probabilities show deteriorating initial schedules when delays occur more frequently. Improvements of the robust schedules, i.e., optimized with regard to the max–min objective, compared with non-robust schedules are more remarkable for the scenario when both kinds of disturbances occur rather than when either exclusively internal or external delays are simulated. This can be seen in the steeper slopes of the curves in Fig. 10 than the slopes from Figs. 8 and 9. Note that the different graphics apply varying scaling factors and bounds on the y-axis.

Spotter schedules optimized with regard to the max $-$ min objective can effectively protect the terminal from dramatic cascades of delays. Even for the most extreme scenario, i.e., when both kinds of disturbances occur with the largest probability and the longest delay duration ($p(h^+) = p(a^+) = 0.2$, and $\max(h^+) = \max(a^+) = 60$), the average weighted lateness is still reasonable when spotter schedules are optimized according to the max–min objective: AWL $\leq$ 44. Thus, in spite of the disturbances, external trucks can be processed almost always on time as planned (TRL $\leq$ 15 min) and our bottleneck resource, the dock doors, do not suffer considerably from delayed jobs (DRL $\leq$ 63 min). Non-robust random solutions, however, lead to considerable delays. For the same scenarios, our lateness measures exceed AWL $\geq$ 277, TRL $\geq$ 81, and DRL $\geq$ 376. In other words, the average waiting time of trucks can be reduced by about 80% (from more than 81 min to less than 15 min) and the average waiting time of a dock door can be reduced by more than 83% (from more than 376 min to less than 63 min) if instead of non-robust schedules a suited robustness measure (i.e., the max–min objective) is applied.

Moreover, we explore the impact of the spotter fleet size m on robustness. In particular, we observe the average values of our three performance measures when executing the spotter schedules derived for the original instances P on the disturbed instances P' with different fleet sizes. The aggregate results are displayed in Fig. 11. We would expect additional spotters to make it easier to increase buffer times since there is more flexibility to shift jobs among the vehicles. And indeed, this is what the results indicate. The decreasing curves of all three charts show an improvement of solution quality (i.e., less lateness) when the spotter fleet size grows. Note that this observation is clearly visible for the robust schedules (i.e., max $-$ min and max $-$ $\sum$), while the spotter fleet sizes seem to have less impact on non-robust (i.e.,

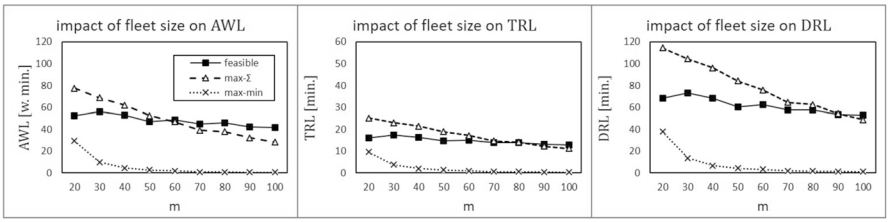


Fig. 11 Impact of spotter fleet size on solution quality

random) schedules. These results can be helpful from a managerial viewpoint at the tactical planning level when deciding on the fleet size. For larger trailer yards, which require a greater spotter fleet anyway, the additional flexibility of swapping jobs among spotters enables sufficient protection against disturbances (when planning with the right robustness objective). For smaller yards and smaller spotter fleets, the pooling effect is much smaller, so that additional stand-by spotters should be procured to support the permanent fleet during peak-hours.

To get a better idea of how objectives max $-\sum$ and max $-$ min are interrelated, we maximize the weighted sum of buffer times for a given aspiration level of $F^{\min}$. Specifically, we first forbid all solutions with an $F^{\min}$ value of less than the given aspiration level, deleting the matching edges from the corresponding graph as described in Sect. 3.2. Then, we solve the RSSP instance with regard to our max $-\sum$ objective.

To illustrate this approach, we solve a random RSSP instance with $n = 500$ jobs, $m = 150$ spotters, and $D = 50$ dock doors. The aspiration level $F^{\min}$ is iteratively increased from 0 by 10, until it reaches the point where the RSSP instance cannot be solved to feasibility. Further, we simulate internal and external disturbances with our simulation model using the parameters $p(h^+) = p(a^+) = 0.2$, and $\max(h^+) = \max(a^+) = 60$ and evaluate robustness by measuring AWL, DRL, and TRL.

Figure 12 visualizes the Pareto frontier for the RSSP instance (i.e., optimal objective values F^{sum} for given aspiration levels $F^{\min}$) and the robustness measures of each Pareto optimal solution. As expected, F^{sum} and $F^{\min}$ are inversely correlated: the higher $F^{\min}$, the lower F^{sum} because a high minimum buffer constrains the feasible search space among which to pick the best F^{sum} solution. However, although a

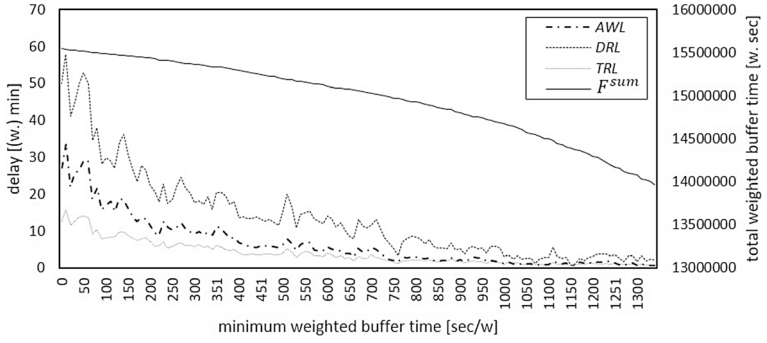


Fig. 12 Simulation results using max $-\sum$ objective with given aspiration level for $F^{\min}$

higher value of $F^{\min}$ leads to lower (i.e., worse) F^{sum} values, it promises better scores on each of three performance measures AWL, TRL, and DRL. These results support the previous findings that our max − min objective yields spotter schedules that are more robust than those of the max − $\sum$ objective.

5 Conclusion and future research

This paper aims to reduce turmoil in large trailer yards resulting in considerable waiting times for the trucks up to several hours in many real-world applications (for examples see Sect. 1). Existing solution approaches primarily address the demand side and aim to spread out truck arrivals over the day, e.g., by applying internet booking platforms. We, however, show that robust internal schedules can successfully avoid that unforeseen delays propagate among successive jobs and considerable waiting times accumulate. We specifically address the assignment of transport jobs (i.e., semitrailers to be moved between the trailer yard and the dock doors of a terminal) to a fleet of spotters. By introducing time buffers among successive jobs assigned the same vehicle, our computational study shows that waiting times of trucks and idle times at the dock doors of the terminal can be reduced considerably. Within a simulation study, especially the max–min objective, which maximizes the minimum time buffer among all successive job pairs, is shown to successfully protect schedules against unforeseen delays. We present a solution procedure for the resulting robust optimization problem that is solvable in polynomial time so that even for very large trailer yards and long planning horizons optimal solutions can be obtained quickly.

Our robust solution approaches are not bound to spotter scheduling in trailer yards, so that future research should evaluate other applications of our robust interval scheduling. A systematic benchmark study including multiple applications and alternative robustness measures would be a valid contribution to further promote the integration of time buffers to protect against uncertainty in a simple and straightforward manner. Thanks to the short computational times of the proposed algorithms, they can also be applied in a dynamic environment where unforeseen events require a repeated (re-)planning on rolling horizons. The only adaption of our solution approaches for such an application is that some spotters are still blocked at the beginning of the current planning horizon by jobs not yet completed. While the algorithmic adaption is truly straightforward, future research should evaluate our solution approaches when planning on rolling horizons. Further computational tests should also compare the proposed approaches with online scheduling, stochastic optimization, and alternative robust optimization techniques. Another important future research task may integrate spotter scheduling with truck scheduling in a holistic approach since these two problems are clearly heavily interdependent. Recall that the proposed algorithms solve the spotter scheduling problem in polynomial time and may thus be part of a decomposition scheme.

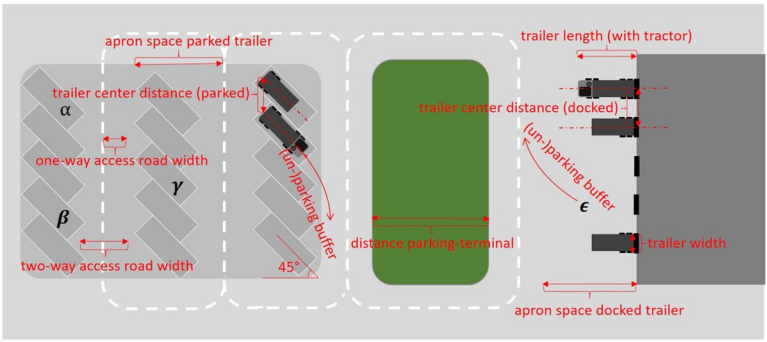
Acknowledgements Open Access funding provided by Projekt DEAL.

Appendix: Distance calculations in the trailer yard

This appendix provides a more detailed explanation of how the distances in the yard within our test instances are calculated. It, thus, complements Sect. 4.1.

Table 5 Input (safety) parameters for trailer yard

Parameter	Values	
	Foot (')/inch (")	Meter
Trailer length (with tractor)	53'	16.1544
Trailer width	96"	2.4384
Trailer center distance (docked)	12'	3.6576
Trailer center distance (parked)	12'	3.6576
One-way access road width	13'	3.9624
Two-way access road width	26'	7.9248
Apron space docked trailer	159'	48.4632
Apron space parked trailer	88'	26.8224
Distance parking-terminal	328.084'	100



$$d_{\alpha\beta} = 2 \times \text{(un-)parking buffer} + 3 \times \text{trailer center distance (parked)}$$
vertical distance; vertical distance

$$d_{\alpha\gamma} = 2 \times \text{(un-)parking buffer} + 6 \times \text{trailer center distance (parked)} + 1 \times \text{(apron space parked trailer + one-way access road width)}$$
vertical distance; horizontal distance

$$d_{\alpha\epsilon} = 2 \times \text{(un-)parking buffer} + 5 \times \text{trailer center distance (parked)} + 2 \times \text{(apron space parked trailer + 6} \times \text{one-way road width + distance parking-terminal) + apron space docked trailer-trailer length}$$
vertical distance; horizontal distance

Fig. 13 Yard layout with safety parameters and distance calculations

When calculating distances, we take into consideration dock and yard safety distances, apron spaces between docked and parked trailers, minimum road widths, and parking angles as defined by the widespread dock planning standards elaborated in Nova Technology (2013). Table 5 summarizes the values of the applied safety distances in the yard, which are visualized for an example layout in Fig. 13. We derive the shortest distances between any pairs of positions in the trailer yard as the sum of total horizontal and vertical distances including the (un-)parking buffers. Figure 13 includes some example distance calculations between four selected positions $\alpha, \beta, \gamma, \epsilon \in \{D \cup \Pi\}$.

References

Aytug H, Lawley MA, McKay K, Mohan S, Uzsoy R (2005) Executing production schedules in the face of uncertainties: a review and some future directions. Eur J Oper Res 161:86–110

Battini D, Boysen N, Emde S (2013) Just-in-time supermarkets for part supply in the automobile industry. J Manag Control 24:209–217

Ben-Tal A, Nemirovski A (2002) Robust optimization methodology and applications. Math Program Ser B 92:453–480

Ben-Tal A, Nemirovski A (2008) Selected topics in robust convex optimization. Math Program Ser B 112:125–158

Berghman L, Leus R (2015) Practical solutions for a dock assignment problem with trailer transportation. Eur J Oper Res 246:787–799

Berghman L, Leus R, Spieksma FC (2014) Optimal solutions for a dock assignment problem with trailer transportation. Ann Oper Res 213:3–25

Bodnar P, de Koster R, Azadeh K (2015) Scheduling trucks in a cross-dock with mixed service mode dock doors. Transp Sci 51:112–131

Boysen N, Fliedner M (2010) Cross dock scheduling: classification, literature review and research agenda. Omega 38:413–422

Boysen N, Fedtke S, Weidinger F (2017) Truck scheduling in the postal service industry. Transp Sci 51:723–736

Briskorn D, Leung J, Pinedo M (2011) Robust scheduling on a single machine using time buffers. IIE Trans 43:383–398

Burkard R, Dell'Amico M, Martello S (2009) Assignment problems. Society for Industrial and Applied Mathematics, Philadelphia

Descartes (2019) Descartes dock appointment scheduling software. https://www.descartes.com/documents/descartes-dock-appointment-scheduling. Accessed Aug 2019

Herroelen W, Leus R (2004) Robust and reactive project scheduling: a review and classification of procedures. Int J Prod Res 42:1599–1620

Hall NG, Posner ME (2001) Generating experimental data for computational testing with machine scheduling applications. Oper Res 49:854–865

Jonker R, Volgenant A (1987) A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing 38:325–340

Karaenke P, Bichler M, Minner S (2019) Coordination is hard: electronic market mechanisms for increased efficiency in transportation logistics. Manag Sci 65:5884–5900

Kolen AW, Lenstra JK, Papadimitriou CH, Spieksma FC (2007) Interval scheduling: a survey. Nav Res Log 54:530–543

Kouvelis P, Yu G (1997) Robust discrete optimization and its applications. Kluwer Academic Publishers, Dordrecht

Ladier AL, Alpan G (2016) Cross-docking operations: current research versus industry practice. Omega 62:145–162

Leus R, Herroelen W (2007) Scheduling for stability in single-machine production systems. J Sched 10:223–235

Nova Technology International (2013) Dock planning standards. https://www.novalocks.com/wp-content/uploads/Dock-Planning-Standards-Guide.pdf. Accessed Aug 2019

Ou J, Hsu VN, Li CL (2010) Scheduling truck arrivals at an air cargo terminal. Prod Oper Manag 19:83–97

Statista (2018) Transportleistung des Straßengüterverkehrs in der Europäischen Union von 1995 bis 2016 (in Milliarden Tonnenkilometer) (in German). https://de.statista.com/statistik/daten/studie/282301/umfrage/transportleistung-des-strassengueterverkehrs-in-der-eu/. Accessed Aug 2019

Tadumadze G, Boysen N, Emde S, Weidinger W (2019) Integrated truck and workforce scheduling to accelerate the unloading of trucks. Eur J Oper Res 278:343–362

Tadumadze G, Emde S, Diefenbach H (2020) Exact and heuristic algorithms for scheduling jobs with time windows on unrelated parallel machines. OR Spect 42:461–497

Van de Vonder S, Demeulemeester E, Herroelen W, Leus R (2005) The use of buffers in project management: the trade-off between stability and makespan. Int J Prod Econ 97:227–240

VRS (2011) Problemzone Rampe. Verkehrsrundschau 24/2011, 24 (in German)

VRS (2012) Nadelöhr Laderampe. Verkehrsrundschau 17/2012, 20–23 (in German)

Yano CA, Bozer Y, Kamoun M (1998) Optimizing dock configuration and staffing in decentralized receiving. IIE Trans 30:657–668