An Exact Solution Approach for Scheduling Cooperative Gantry Cranes*

Dominik Kress[a,*], Jan Dornseifer[b], Florian Jaehn[c]

[a] *University of Siegen, Management Information Science, Kohlbettstr. 15, 57068 Siegen, Germany*
[b] *Tebis ProLeiS GmbH, Marburger Str. 3a, 57339 Erndtebrück, Germany*
[c] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Institute for Management Science and Operations Research, Holstenhofweg 85, 22043 Hamburg, Germany*

Abstract

We consider a scheduling problem for two gantry cranes moving on the same rails at a single storage block. Containers originating at the seaside have to be stored in the block and containers that are already stored in the storage area at the beginning of the planning horizon have to be delivered to the landside handover point within given time windows. Most commonly in seaport operations, the berthing time of vessels is to be minimized. Thus, the objective considered in this article is to minimize the makespan of seaside container processing while guaranteeing on-time processing of landside containers and while considering non-crossing constraints among cranes. We allow preemption of seaside container processing. This means that one crane may move a seaside container to an intermediate storage slot, and the other crane takes it to its designated position. This has previously been shown to be an effective method of reducing the makespan when compared to classical approaches. We present a dynamic programming (DP) algorithm and a related beam search heuristic. The DP method makes use of bounding techniques and applies dominance properties of optimal solutions. In computational tests, we show that the DP approach clearly outperforms CPLEX and that it is able to quickly solve instances with real-world yard settings. The beam search heuristic is shown to be capable of quickly improving solutions of heuristic approaches that have previously been introduced in the literature. This allows both algorithms to be applied in real-world online settings, where container data is revealed incrementally.

Keywords: Scheduling, Container logistics, Seaport logistics, Twin cranes, Crane scheduling

1. Introduction

Standardized containers play a key role in the unitload-concept and have thus become an essential part of modern logistics processes over the past decades (Stahlbock & Voß, 2008; Steenken et al., 2004).

*Corresponding author
Email addresses: dominik.kress@uni-siegen.de (Dominik Kress), jan.dornseifer@tebisproleis.com (Jan Dornseifer), florian.jaehn@hsu-hh.de (Florian Jaehn)

* This is an Accepted Manuscript of an article published by Elsevier in the **European Journal of Operational Research** on 6 August 2018, available online: https://doi.org/10.1016/j.ejor.2018.07.043

© 2018. This manuscript version is made available under the CC-BY-NC-ND 4.0 license http://creativecommons.org/licenses/by-nc-nd/4.0/

Large transshipment terminals at seaports handle several million twenty-foot equivalent units (TEU) on an annual basis. The Port of Rotterdam, for example, handled more than 12 million TEU in 2016 (Port of Rotterdam, 2017). It is therefore not surprising that port authorities strive for sophisticated planning approaches based on simulation or optimization techniques in order to stay competitive.

In a typical scenario at a container port, *inbound containers*, i.e. containers arriving by vessel, are unloaded by quay cranes upon the arrival of vessels at the berths. The containers are then transported to large storage areas, also referred to as blocks, by automated or non-automated vehicles. Starting from seaside handover positions in front of the blocks, gantry cranes that span over the storage areas then pick up the containers and transfer them to intermediate positions within the blocks, where they are stored for further processing. Once the containers are requested at the landside, they must be transported to landside handover points from their storage positions within the blocks by the blocks' gantry cranes. Then, again, the containers are handled by vehicles and are finally loaded onto trains or trucks. *Outbound containers*, i.e. containers that arrive by train or truck and have to be loaded onto a vessel, process the same steps in reverse order. In addition to these two types of containers, there exist containers that arrive by vessel and will be loaded onto another vessel, which implies that these containers enter and leave the blocks on the seaside. These latter containers are usually referred to as *transit containers*.

1.1. Related Literature and Contribution of this Article

There exists a vast amount of research articles that deal with operations research challenges arising at container terminals in general and, in particular, regarding the processing of containers at seaports. Comprehensive overviews on the former general perspective are provided by Steenken et al. (2004) and Stahlbock & Voß (2008). A very broad survey of research in the field of ocean container transport is presented by Lee & Song (2017). Literature reviews for berth allocation problems and the scheduling of quay cranes are given by Bierwirth & Meisel (2010, 2015). In this article, however, we focus on optimizing processes at the storage blocks. An excellent review on this topic has recently been given by Carlo et al. (2014) so that we refer the interested reader to this article and the update of related publications regarding the scheduling of gantry cranes within the storage area in Jaehn & Kress (2018).

Some papers have been published very recently. They include Gharehgozli et al. (2017), who use simulation to evaluate cooperative twin cranes that use a "handshake area" for handing containers over. Speer & Fischer (2017) compare different crane configurations within a storage block. Lashkari et al. (2017) focus on to the unloading process of a vessel, but their work on cranes with spreaders that are able to concurrently lift two containers is also relevant for storage blocks. Finally, a classification scheme for crane scheduling with interference constraints, which can, amongst others, be applied to seaport terminals and to rail terminals, is presented by Boysen et al. (2017). Following their scheme, we will consider problem [1D,2,ends — pmtn, mv^{X},r^i,δ^i,pos — $C^{\max}$] in this article. The entries in

this classification scheme refer to the following problem properties: storage positions of containers are considered in one dimension, there are two cranes, and input and output containers only appear on opposite sides of the block. Preemption is allowed, cranes move at constant speed, release times and deadlines are considered, there are predefined initial positions of the cranes, and finally, the objective is to minimize the schedule length (makespan).

More specifically, we focus on a specific gantry crane setting with two cranes spanning over a single storage block. It is referred to as the *twin system* (Kemme, 2012) and is depicted in Figure 1. The

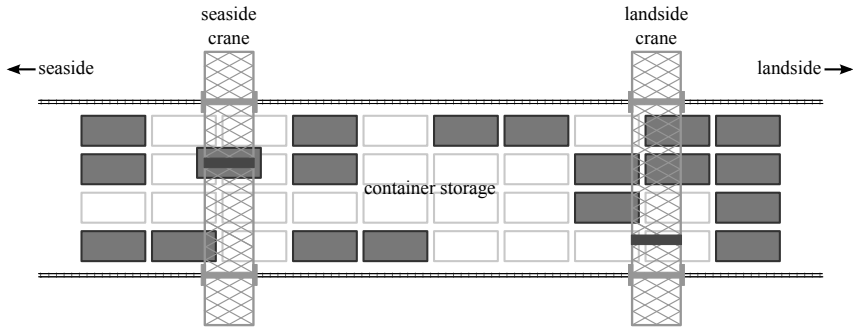


Figure 1: Schematic layout of a twin system (Jaehn & Kress, 2018)

storage area is enclosed by the seaside and landside handover points. The gantry cranes are identical, rail mounted, and share the same tracks for their horizontal movement along the long side of the block. They therefore cannot pass each other, such that we have to take account of non-crossing constraints. One of the cranes, the *seaside crane*, is responsible for serving the seaside and the other one, the *landside crane*, must serve the landside. The cranes are equipped with a *spreader* that allows for lifting and dropping the containers. Usually, during lifting and dropping, no concurrent movements of the container are possible. However, when a crane moves horizontally, the spreader can simultaneously move along the short (vertical) side of the block. Additionally, the block is usually very long but not very wide so that the spreader is typically fast enough to complete its positioning during the crane's horizontal movement. Therefore, these vertical moves will be neglected in our considerations by assuming the slots of the storage block to be arranged along a single straight line (see Section 2 for details), as it is often done in other approaches for container terminals (e.g. Ehleiter & Jaehn, 2016; Kovalyov et al., 2018).

The workload of the cranes varies significantly over time. We focus on critical time periods in which vessels have to be unloaded and the major objective of port authorities is the minimization of dwell times of vessels at the berth. The problem of loading a vessel is of similar nature and the methods proposed here can similarly be applied to the loading problem. However, for ease of description and considering the fact that in European ports a much larger volume of goods is imported than exported, we neglect the loading problem. In the aforementioned peak times, containers that have to be unloaded from a vessel are usually assigned highest priority and the seaside crane of a relevant twin system stores containers in its block nonstop while the landside crane performs other tasks, e.g. delivering inbound containers to the landside handover point. However, as observed by Briskorn et al. (2016), it might

be beneficial to allow the seaside crane to move inbound containers only part of the way to so-called handover storage positions and let the landside crane finish the transportation of these containers to their target slots within the block, i.e. allow the cranes to *cooperate*. The authors restrict their analysis to containers that originate at the seaside and have to be stored in the block (*seaside containers*). When the sequence of seaside containers is given, they refer to this setting as the *preemptive crane scheduling problem with a given unloading sequence* (PCSP-S). In Jaehn & Kress (2018), this analysis is extended by including additional hinterland traffic, i.e. containers that are already stored in the block at the beginning of the planning horizon and that have to be delivered to the landside handover point by the landside crane within given time windows (*landside containers*). In line with Briskorn et al. (2016), the authors refer to this problem as the PCSP-SL, where the L indicates the existence of landside containers. The authors show that finding a feasible solution to an instance of PCSP-SL is NP-hard in the strong sense. Furthermore, a procedure to determine lower bounds and two heuristic algorithms are presented. These heuristics extend an approach that was introduced by Briskorn et al. (2016) and that makes use of the bucket brigade principle, which we illustrate in detail in Section 2.2. Based on these algorithms and an extensive computational study, Jaehn & Kress (2018) show that cooperation might reduce the makespan by more than 20% when compared to the situation in which seaside containers are exclusively served by the seaside crane. The authors conclude that allowing cooperation of gantry cranes may result in significant time savings when unloading vessels and should therefore be considered by terminal operators when determining crane schedules.

Based on the observation that CPLEX performs very poorly for small instances of PCSP-S (see Briskorn et al., 2016), it is the aim of this article to extend the aforementioned studies by presenting a dynamic programming (DP) approach for PCSP-SL. Besides making use of bounding techniques, this DP applies dominance properties of optimal solutions that we will introduce in this article. If the resulting exact approach is able to solve small to medium sized instances to optimality, this will enable us to better evaluate the quality of heuristic approaches. Furthermore, this might allow the DP to be applied in real-world online settings of PCSP-SL, where container data is revealed incrementally. In addition to our DP, we will contribute to the literature by introducing and evaluating a beam search (BS) heuristic for PCSP-SL.

1.2. Overview of this Article

The remainder of this article is organized as follows. In Section 2, we will describe the problem setting and, to keep the paper self-contained, briefly summarize the lower bounds and heuristics introduced by Jaehn & Kress (2018), following their line of arguments. Next, in Section 3, we will introduce dominance properties of optimal solutions of PCSP-SL. They will turn out useful when designing our exact dynamic programming approach in Section 4. Computational results are presented in Section 5. The paper closes with a conclusion in Section 6.

2. Preliminaries

We denote the cranes c of the considered twin system by $c = w$ in case of the *seaside crane* and $c = l$ in case of the *landside crane*. As motivated in Section 1, the vertical extension of the storage block is assumed to be one, and the *slots* (storage positions) s of the block are numbered consecutively from 0 to $S + 1$ (see Figure 2). Slots 0 and $S + 1$ represent the seaside and landside handover points (input/output, I/O, points), respectively. The cranes must not be located in the same slot at any time instant. For ease of description, we do not consider safety distances between the cranes. Note, however, that they can be incorporated into the presented methods. Moreover, we do not consider stacking constraints and we do not take limited storage capacities of the slots into account, but assume that the capacities are sufficiently large throughout the planning horizon.

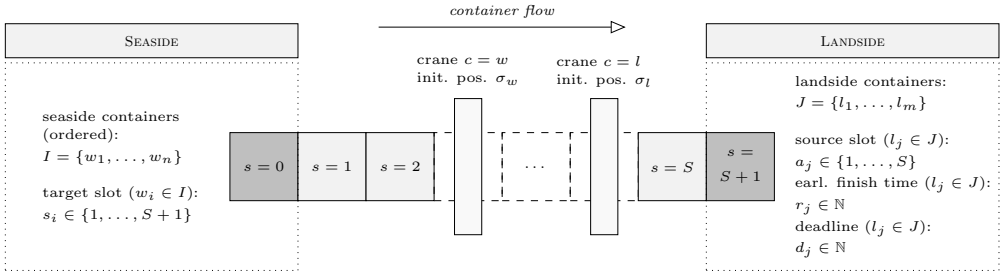


Figure 2: Problem setting and notation (Jaehn & Kress, 2018)

We assume that the considered time horizon is divided into a finite number of intervals $[t - 1, t]$, $t = 1, 2, \ldots$, of equal length. Interval $[t - 1, t]$, $t = 1, 2, \ldots$, is also referred to as time slot t. The length of a time slot is referred to as a time unit and is defined to correspond to the time needed by a crane to move from a given slot to a neighboring slot. All time parameters in the remainder of this paper are assumed to be integral multiples of a time unit and are therefore specified by natural numbers. The number of time units required to lift (pick up) or drop a container while not moving between slots is denoted by $p \in \mathbb{N}$.

We restrict ourselves to considering containers that enter the storage block on the seaside and leave the block on the landside, i.e. we consider an unidirectional flow of inbound containers, with a total of n *seaside containers* originating at the seaside (slot 0) and m *landside containers* having to be moved to the landside (slot $S + 1$). The sets of seaside containers and landside containers are denoted by $I = \{w_1, \ldots, w_n\}$ and $J = \{l_1, \ldots, l_m\}$, respectively. The set I is assumed to be ordered with respect to a given *pick-up sequence*, i.e. container $w_i \in I$ must be picked up and moved to a slot $s > 0$ by the seaside crane before container $w_j \in I \setminus \{w_i\}$, $j > i$, can be picked up. Each seaside container $w_i \in I$ is associated to one parameter, namely a *target slot* $s_i \in \{1, \ldots, S + 1\}$, where it must be dropped. Each landside container $l_j \in J$ has multiple associated parameters:

- A *source slot* $a_j \in \{1, \ldots, S\}$ where it originates from, i.e. where it must be lifted.
- An earliest time slot $r_j \in \mathbb{N}$ (*earliest finish time*) at the end of which it may be dropped in slot

$S+1$. The landside crane may start processing a landside container before its earliest finish time at the cost of potentially having to wait at the landside handover point, e.g. because the receiving truck has not yet arrived.

- A latest time slot $d_j \in \mathbb{N}$ (*deadline*) at the end of which it must be dropped in slot $S+1$.

The latter two parameters r_j and d_j define the *time window* $[r_j, d_j]$ of landside container $l_j \in J$.

We allow preemption of seaside container processing, i.e. the seaside crane may move a seaside container part of its way, while the landside crane takes it to its target slot. However, we assume that each seaside container may be processed at most once by each crane. A preempted container is referred to as a *handover container*. Landside containers may only be processed by the landside crane.

A *crane schedule* is defined by the positions $x_{c,t}$ of both cranes $c \in \{w, l\}$ at all time instants t of the time horizon as well as the cranes' operations in the respective time slots. The initial locations of the cranes are denoted by $\sigma_w = x_{w,0} \in \{0, \ldots, S\}$ and $\sigma_l = x_{l,0} \in \{1, \ldots, S+1\}$ with $\sigma_w < \sigma_l$. Each crane performs exactly one operation in a given time slot. Table 1 illustrates the potential crane operations as well as the notation used to denote these operations in the remainder of this paper.

Table 1: Potential operations of crane $c \in \{w, l\}$ in interval $[t-1, t]$

operation	symbol	position	description
move left	$\leftarrow$	$x_{c,t} = x_{c,t-1} - 1$	c is moving left
move right	$\rightarrow$	$x_{c,t} = x_{c,t-1} + 1$	c is moving right
lift	$\uparrow$	$x_{c,t} = x_{c,t-1}$	c is in the process of lifting a container
drop	$\downarrow$	$x_{c,t} = x_{c,t-1}$	c is in the process of dropping a container
wait	$\circ$	$x_{c,t} = x_{c,t-1}$	c is not moving and neither lifting nor dropping a container

Given a crane schedule, the position (slot) of seaside container $w_i \in I$ in the storage area at time instant t is denoted by $y_{i,t}$. Similarly, $z_{j,t}$ denotes the position of landside container $l_j \in J$ at time instant t. Moreover, a crane is referred to as *loaded* (*unloaded*) at time instants $t, \ldots, t'$ if it lifts (drops) a container in time slots $t-p+1$ to t and drops (lifts) this very (the next) container in time slots $t'+1$ to $t'+p$. Additionally, a crane is referred to as unloaded at the beginning of the planning horizon, i.e. from time instant 0 until it starts lifting the first container. In all other time instants, the crane is neither considered being loaded nor unloaded. A crane that is loaded (unloaded) at time instants $t-1$ and t, $t > 0$, is said to be loaded (unloaded) in time slot t.

As outlined above, port authorities mainly focus on minimizing dwell times of vessels, so that we aim at finding a feasible crane schedule that minimizes the *makespan* $C_{\max}$ of seaside container processing (seaside makespan), i.e. the earliest time instant at which all seaside containers have been dropped in their target slots. With respect to the landside containers, we assume that a crane schedule is feasible if all landside containers with deadlines less or equal to $C_{\max}$, i.e. all containers of the set $J^{C_{\max}} := \{l_j \in J | d_j \leq C_{\max}\}$, are dropped off on time. Furthermore, as infeasibility should not occur right after $C_{\max}$ in practical applications, a solution is only considered being feasible if a next landside container $l_{j^{next}}$, $j^{next} \in \arg\min_{l_j \in J \setminus J^{C_{\max}}} d_j$, can be dropped off on time.

A mixed-integer program for PCSP-SL is presented in Appendix A.

2.1. Lower Bounds

For a given number h of handover containers, Jaehn & Kress (2018) introduce two lower bounds on the seaside makespan of an instance of PCSP-SL. $LB(h)$ is based on approximating the total number of lifts and drops as well as the total distance that must be covered by *both* cranes. $LB^{(sea)}(h)$, on the other hand, does not take the landside crane into account. It solely bounds the seaside crane's workload from below. $LB(h)$ is increasing in h, i.e. $LB(0)$ is a general lower bound, and $LB^{(sea)}(h)$ is non-increasing in h, i.e. $LB^{(sea)}(n)$ is a general lower bound. Given these bounds, the authors then present an algorithm that determines a general lower bound based on the following idea. $LB(h)$ is loose if $LB^{(sea)}(h)$ is larger than $LB(h)$. Moreover, $LB(h+1)$ is a general lower bound if $LB^{(sea)}(h)$ is larger than $LB(h+1)$. Hence, a lower bound can be computed by successively incrementing the number h of handover containers until either $LB^{(sea)}(h) \leq LB(h)$, so that $LB(h)$ is a lower bound, or solely $LB^{(sea)}(h) \leq LB(h+1)$, so that $LB^{(sea)}(h)$ is a lower bound. Additionally, if $s_n \neq S+1$ in an instance of PCSP-SL, then there exists an optimal solution with at most $n-1$ handover containers. These ideas are summarized in the following algorithm (as presented by Jaehn & Kress, 2018).

Algorithm 1 (Determine Lower Bound)

0. Initialization: Set $LB := 0$, $LB^* := 0$, and $h := 0$.

1. Bound on h: If $h < n-1$ and $LB(h+1) < LB^{(sea)}(h)$, then $h := h+1$ and go to Step 1.
If $LB(h) < LB^{(sea)}(h)$, then $LB := LB^{(sea)}(h)$, else $LB := LB(h)$

2. Iteration: If $LB^* \geq LB$, then stop. Else, set $LB^* := LB$, $h := 0$ and go to Step 1.

Note that one can easily construct a modified version of Algorithm 1 that allows being applied to partial solutions in which the crane schedule is fixed up to some given time instant.

2.2. Bucket Brigade Heuristics

Jaehn & Kress (2018) present two heuristic approaches for PCSP-SL. Both approaches extend an algorithm for PCSP-S that was introduced by Briskorn et al. (2016) and makes use of the *bucket brigade* principle: The seaside crane moves left to slot 0 and picks up the next seaside container (if existent) whenever it is unloaded. When being loaded, it moves right and either drops the container upon having reached its target slot, or, if the non-crossing constraints prohibit further moves to the right, it drops the container for immediate handover. If the landside crane is unloaded and not lifting or dropping a container, it moves left until it meets the seaside crane. It then potentially waits and afterwards moves left to pick up the handover container that was dropped by the seaside crane because of having met the landside crane. The landside crane then delivers this container to its target slot.

Both extensions of the bucket brigade principle by Jaehn & Kress (2018) dynamically update the set of landside containers that must still be served within the planning horizon by applying the lower bound determined by Algorithm 1. They differ in their strategy of how to handle those containers.

In the *simple bucket brigade* (SB) approach, the decision of whether the landside crane continues bucket brigade, i.e. potentially processes a handover container, or interrupts bucket brigade in order to process a landside container, is initiated when the landside crane has dropped a container. If there exists a landside container that can be served without having to wait in slot $S + 1$, this container is prioritized over any seaside container. Once SB reaches an infeasible partial solution with a late landside container, it applies a backtracking strategy that triggers an earlier abortion of the bucket brigade mode to potentially be able to serve this landside container within its time window.

One of the drawbacks of SB is the fact that the prioritization of landside containers may result in solutions with many unloaded moves of the landside crane. In a second variant of the bucket brigade principle, referred to as the *bundling bucket brigade* (BB) procedure, Jaehn & Kress (2018) therefore bundle landside containers for sequential processing. To do so, BB determines a landside container with earliest deadline among the containers not yet served and initializes a bundle with this container. It assumes that this container is dropped off in slot $S + 1$ right at its deadline. BB then adds all landside containers to the bundle whose earliest finish times allow the landside crane to process the bundle without waiting in slot $S + 1$ and without processing seaside containers in between. Next, a time window for the bundle is computed. It includes all time instants at which the processing of the bundle can be started so that the landside crane can sequentially process all containers of the bundle without having to wait at the landside handover point. The bundle and its time window are potentially modified when the landside crane finishes a dropping operation. The landside crane then starts serving the bundle if this is possible within the time window or, if this is not the case, continues bucket brigade. As before, when an infeasible partial solution is reached, a backtracking procedure is applied.

Additionally, as the landside crane cannot always process handover containers immediately after they have been dropped off by the seaside crane when landside containers are present, SB and BB apply specific rules on how to handle those containers (for details, see Jaehn & Kress, 2018).

3. Dominance Properties of Optimal Solutions

Even though there might exist more than one optimal solution to an instance of PCSP-SL, it is sufficient to search for just one optimal solution. In the following, we will present properties of optimal solutions that can be exploited to guide an algorithm that is restricted to exclusively search for this solution. These properties, which might be rather intuitive at first glance, are of particular interest, as optimal solutions might show counterintuitive characteristics. For example, there exist instances in which the landside crane moves left while being loaded in every optimal solution (Jaehn & Kress, 2018).

For the sake of brevity, the proofs of the dominance properties are presented in Appendix B. Recall that a crane that is in the process of lifting or dropping is neither considered to be loaded nor unloaded.

3.1. Dominance Properties for the Seaside Crane

The first three properties allow for focusing on optimal solutions in which the seaside crane stays as close to its handover area as possible.

Consider an arbitrary instance of PCSP-SL with at least one feasible schedule. There always exists an optimal solution with the following properties regarding the seaside crane:

Property 1. *In every time slot t in which the seaside crane is unloaded, it either moves left ($x_{w,t-1} = x_{w,t} + 1$) or, if it is in slot 0 and has already processed all seaside containers, i.e. $x_{w,t-1} = 0$ and $y_{i,t-1} > 0$ for all $i \in \{1, ..., n\}$, it waits ($x_{w,t-1} = x_{w,t}$).*

Property 2. *If the seaside crane is loaded in time slot t it does not move left, i.e. $x_{w,t-1} \leq x_{w,t}$.*

Property 3. *If the seaside crane is loaded with some container $w_i \in I$ and located at the container's target slot at some time instant $t-1$ ($x_{w,t-1} = s_i$), it immediately drops the container in time slots t to $t-1+p$.*

It follows from Property 3 that w.l.o.g. we may assume that dropping operations are never preceded by waiting and that handover containers are always dropped in a slot left of their target slot.

3.2. Dominance Properties for the Landside Crane

We now turn our attention to dominance properties for the landside crane. Again, we focus on solutions in which the landside crane is located as close to the seaside (i.e. it stays 'left') as possible. We start by considering the case of the landside crane being unloaded.

Property 4. *Let OPT be an optimal solution to an instance of PCSP-SL and let t be a time slot in OPT with the landside crane being unloaded and with*

1. *$x_{l,t-1} > x_{w,t}$ (moving left is not prohibited by non-crossing constraint of cranes),*
2. *$x_{l,t-1} > y_{i,t-1}$ for all $w_i \in I$ with $y_{i,t-1} \neq s_i$ (all seaside containers that have not yet reached their target slot are located to the left of the landside crane),*
3. *$x_{l,t-1} > S + 1 + 2p + t - r_j$ for all $l_j \in J$ with $z_{j,t-1} \neq S + 1$ (all landside containers that have not yet been dropped by the landside crane may reach slot $S+1$ at their respective earliest finish time, even if the landside crane were to move left).*

Then there exists an optimal solution in which the landside crane moves left in period t ($x_{l,t-1} = x_{l,t}+1$).

Now consider an arbitrary instance of PCSP-SL with at least one feasible schedule. If the landside crane is loaded with a landside container, we may assume that it moves right until it reaches the handover area, i.e. there always exists an optimal solution with the following property:

Property 5. *If the landside crane is loaded with a landside container $l_j \in J$ at time instant $t-1$, it either moves right in time slot t ($x_{l,t-1} = x_{l,t} - 1$) or, if $x_{l,t-1} = S + 1$ and $r_j > t - 1 + p$, it waits in*

time slot t ($x_{l,t-1} = x_{l,t}$), or, if $x_{l,t-1} = S + 1$ and $r_j \leq t - 1 + p$, it starts dropping the container in time slot t.

As mentioned above, there exist instances in which the landside moves left while being loaded in every optimal solution. However, we can restrict ourselves to assuming that this only happens if the landside crane gives way to the seaside crane because, in case of the existence of at least one feasible schedule, there always exists an optimal solution with the following properties:

Property 6. *For every time slot t in which the landside crane is loaded with a seaside container $w_i \in I$ and in which it waits ($x_{l,t-1} = x_{l,t}$), both cranes are in neighboring slots at time instant t, i.e. $x_{w,t} = x_{l,t} - 1$.*

Property 7. *For every time slot t in which the landside crane is loaded with a seaside container $w_i \in I$ and in which it moves left ($x_{l,t-1} = x_{l,t} + 1$), both cranes are in neighboring slots at time instants t and $t - 1$, i.e. $x_{w,t} = x_{l,t} - 1$ and $x_{w,t-1} = x_{l,t-1} - 1$.*

3.3. Dominance Properties for Both Cranes

Consider an arbitrary instance of PCSP-SL with at least one feasible schedule. There always exists an optimal solution with the following properties:

Property 8. *None of the cranes' lifting or dropping operations are interrupted once they have started, i.e. if a crane is lifting a container in time slot t and if it is not loaded at time instant t, then it continues lifting this very container in time slot $t + 1$.*

Property 9. *None of the cranes' lifting operations are preceded by waiting, i.e. if a crane is lifting a container in time slot t, then it has not been waiting in time slot $t - 1$.*

3.4. Summary of the Properties

Consider a feasible schedule of an instance of PCSP-SL and denote a specific operation of the set $\{\uparrow, \downarrow, \leftarrow, \rightarrow, \circ\}$ that is performed by a crane c in a given time slot $t > 0$ of this schedule by $o_{c,t}$. Furthermore, refer to the first succeeding operation of c that differs from the one in t within the schedule as the *follow-up operation* of $o_{c,t}$. For each crane, Table 2 depicts all pairs of follow-up operations that are prohibited (marked by an asterisk) in optimal solutions that are not dominated by the properties presented above. As, by definition, follow-up operations must differ from their preceding operations, dashes mark pairs of identical operations which are not relevant for the table. Note that, in addition to the above dominance properties, Table 2 includes some straightforward optimality criteria and restrictions, e.g. the fact that a crane never drops a container in the same slot where it has been lifted or that a loaded crane cannot lift a container. The table can of course be applied to PCSP-S as well.

Table 2: Prohibited follow-up operations for PCSP-SL

operation	seaside crane					landside crane				
	↑	↓	←	→	∘	↑	↓	←	→	∘
pick-up (↑)	-	*	*			-	*	*		*
drop-off (↓)	*	-		*	*		-			
move left (←)										
loaded	*	*	-	*	*	*		-	*	*
unloaded		*	-	*			*	-		
move right (→)										
loaded	*		*	-		*		*[1]	-	
unloaded	*	*	*	-	*		*		-	
wait (∘)										
loaded	*	*	*		-	*		*[1]	*	-
unloaded	*	*	*	*	-	*	*			-

[1]: in case of landside containers

4. Bounded Dynamic Programming

The set of feasible solutions is drastically reduced if it is restricted to the solutions following the dominance properties summarized in Table 2. We will make use of this reduction by presenting a corresponding DP procedure that generates an optimal solution by successively assigning containers and corresponding (potentially temporary) destination slots to the cranes. Within this algorithm, the dominance properties will also be used for dissolving potential crane interferences in partial solutions. Before presenting the details of the algorithm, we will illustrate the underlying definition of states and the generation of the DP graph. We will then outline our methods of evaluating and fathoming states. Finally, we will present a heuristic beam search procedure based on our DP algorithm.

4.1. Definition of States

Our DP procedure is based on successively assigning *jobs* to the cranes. In case of the seaside crane, a job corresponds to the next available (or the currently processed) seaside container w_i as well as a destination slot, i.e. any potential handover slot, denoted by h_i, or the target slot s_i. For the landside crane, a job can be any landside container l_j that has not yet been dropped in slot $S+1$ or any seaside container w_i that has not yet reached its target slot. In the latter case, the origin slot of w_i is the handover slot h_i in which the seaside crane has dropped the container. If the container has not yet been dropped by the seaside crane, we set $h_i = 0$ for notational convenience. Note that we use the term *origin slot* to generalize the terms seaside I/O, handover slot, and source slot. Similarly, *destination slot* generalizes the terms landside I/O, handover slot, and target slot. This is summarized in Table 3.

Table 3: Origin and destination slots of containers

	origin slot		destination slot	
	seaside crane	landside crane	seaside crane	landside crane
seaside container	seaside I/O	handover slot	handover slot, target slot	target slot
landside container	-	source slot	-	landside I/O

For our definition of states we make use of the fact that, once the next job to be processed by each

of the cranes has been assigned and when a priority rule in case of an interference is given, the cranes' movements are easily computable based on the assumption that the cranes process their jobs as fast as possible and by making use of the dominance properties of Section 3, until one of the cranes finishes processing its assigned job. Hence, a state describes a situation in which at least one of the cranes has just completed a job, i.e. has dropped the corresponding container in the destination slot, and is therefore ready to process a new container. In this context, we define a crane to be *active* at time instant t, if at least one of the following is true:

- The crane is loaded at time instant t.
- The crane is lifting a container in time slot t.
- The crane is dropping a container in time slot t without being unloaded at time instant t.

This gives rise to our definition of a state, which we denote by

$$(t, x_{w,t}, x_{l,t}, \mathit{active}, \mathit{container}, c^w, \mathit{list}^h, \mathit{list}^l),$$

where, in addition to the notation introduced in Section 2, we define:

$t \in \{0, 1, \ldots\}$: current time instant,

$\mathit{active} \in \{\emptyset, w, l\}$: currently active crane,

$\mathit{container} \in I \times \{\leftarrow, \rightarrow, \downarrow, \uparrow, \circ\} \times \{0, \ldots, p-1\}$ (if $\mathit{active} = w$): container processed by the seaside crane in time slot t, including crane operation and counter on how many time units are left until operation is finished (zero in case of operations $\leftarrow$, $\rightarrow$ and $\circ$),

$\mathit{container} \in I \cup J \times \{\leftarrow, \rightarrow, \downarrow, \uparrow, \circ\} \times \{0, \ldots, p-1\}$ (if $\mathit{active} = l$): container processed by the landside crane in time slot t, including crane operation and counter on how many time units are left until operation is finished (zero in case of operations $\leftarrow$, $\rightarrow$ and $\circ$),

$c^w \in \{0, \ldots, n\}$: counter on the number of seaside containers, the processing of which has not yet been started by the seaside crane,

$\mathit{list}^h \subseteq I \times \{1, \ldots, S-1\}$: list of handover containers that have been dropped off by the seaside crane but the processing of which has not yet been started by the landside crane, including their positions,

$\mathit{list}^l \subseteq J$: list of landside containers, the processing of which has not yet been started by the landside crane.

If no crane is active at time instant t, we define $\mathit{container} = \emptyset$. Hence, the initial state is denoted by $(0, \sigma_w, \sigma_l, \emptyset, \emptyset, n, \emptyset, J)$. Moreover, note that *container* never contains a drop operation with remaining time zero because an unloaded crane is not active by definition.

4.2. The DP Graph

Given the initial state, w_1 is the first container that has to be processed by the seaside crane. Because of Property 3, we can assume w.l.o.g. that this container will be dropped off at s_1 or at any slot further to the left for handover, which results in a number of potential job assignments for the seaside crane. The landside crane's first job may be any seaside or landside container that has to be dropped at its destination slot. For every resulting pair of job assignments to the cranes (which, by definition, include or imply destination slots), at most two new states are derived in order to construct the DP graph. If no interference arises, there is only one state transition. If there is an interference, two states are derived; one giving priority to the landside crane and one giving priority to the seaside crane. In both cases, each resulting state represents the situation in which the first one of the cranes finishes processing its current job at the earliest possible time instant. Note that the landside crane will never be prioritized if its next job corresponds to a handover container that is currently handled or that has not yet been processed by the seaside crane.

4.2.1. On the Use of the Dominance Properties

Obviously, the potential number of states is extremely large. However, as indicated above, an acceptable number of states is achieved by using the dominance properties of Section 3. Given a state and the next job assignment to each of the cranes, it is then possible to make use of the properties to easily compute time instants at which the cranes arrive at the origin slots, finish lifting, reach their jobs' destination slots, and finish dropping, whereby only a subset of feasible crane movements must be considered. The follow-up state(s) can then be determined based on these computations without losing the ability to determine an optimal solution in the overall process. This is illustrated in Example 1.

Example 1. Consider the state $(t, 2, 3, \emptyset, \emptyset, n-1, \emptyset, \{l_1\})$ of an example instance of PCSP-SL with $S = 6$ and $p = 4$. Assume that we want to generate succeeding states based on assigning w_2, with $s_2 = 1$, to the seaside crane and l_1, with $a_1 = 2$ and $r_1 = t+14$, to the landside crane. Obviously, the corresponding destination slots must be slot 1 and slot 7, respectively. Hence, there is no interference between the cranes when immediately processing these jobs in accordance with the dominance properties. By applying Properties 1, 8 and 9, we can assume that the seaside crane directly moves to the seaside handover point and starts lifting w_2 immediately. It is then loaded with w_2 at time instant $t+x_{w,t}+p = t+6$. Similarly, based on Table 2, especially Properties 8 and 9, we may assume that the landside crane moves to a_1 and immediately lifts l_1. It is therefore loaded at time instant $t+(x_{l,t}-a_1)+p = t+5$. Now, by applying Properties 3 and 8, we can assume that the seaside crane will finish processing its assigned job by directly moving to slot s_2 and dropping the container, which is finished at $(t+6)+s_2+p = t+11$. Taking Properties 5 and 8 into account, we may furthermore assume that the landside crane immediately moves to slot $S+1$, arrives at time instant $(t+5)+(S+1-a_1) = t+10$, and drops l_1, which is finished at time instant $t + 14$. Hence, when comparing the times at which the cranes finish their assigned

jobs, the above job assignment results in one follow-up state with the landside crane being active, i.e. $(t+11, 1, S+1, l, (l_1, \downarrow, 3), n-2, \emptyset, \emptyset)$. □

The dominance properties can also be used to assist in detecting and dissolving interferences between the cranes with respect to optimality, which is illustrated in the following example. Details on the detection of interferences are presented in Appendix C.

Example 2. Consider a state $(t, 7, 8, w, (w_i, \downarrow, 3), n-i, \{\dots (w_k, 2), \dots\}, \{\dots\})$ in an example instance of PCSL-SL with $p = 4$. Figure 3 presents an exemplary job assignment in which an interference arises: the seaside crane finishes its current job (drop w_i in slot 7) and the landside crane is assigned a handover container w_k at position 2 that shall be moved to its target slot $s_k = 7$ (case a) or $s_k = 10$ (case b). The interference arises in slot 7. If both cranes were to process their jobs immediately, the seaside crane would finish processing w_i in slot 7 at time instant $t+3$ while the landside crane would enter this slot at $t+1$. As previously mentioned, it is therefore necessary to generate two succeeding

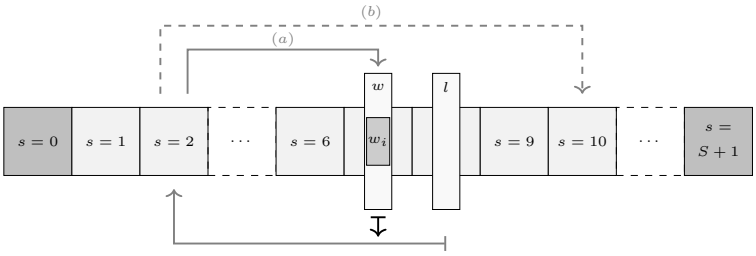


Figure 3: Interference scenario

states based on prioritizing the cranes in order to dissolve the interference. Obviously, if the seaside crane has priority, the landside crane must wait in slot 8 for 3 time units in cases (a) and (b). When prioritizing the landside crane, an intuitive way of dissolving the interference is to make the seaside crane interrupt the dropping operation, lift the container, and move left to slot 1 to give way to the landside crane. This, however, may result in missing an optimal solution in the overall DP procedure because the landside crane would have to wait for one time unit in slot 8 until the seaside crane has finished lifting w_i. Additionally, from a practical perspective, the energy consumption that is induced by making the seaside crane move back and forth between slots 7 and 1 is avoidable. However, by applying Properties 2, 3, and 8, we can assume that the seaside crane starts waiting in slot 1 at time instant $t-(p-3)-(x_{w,t}-1) = t-7$ before it enters slot 2 on its way to drop w_i in slot 7, which allows the landside crane to process its job without waiting. In order to determine the number of periods in which the seaside crane waits, we must distinguish between cases (a) and (b). In case (b), the landside crane drops w_k to the right of slot 7, such that the seaside crane can move to the right as soon as the landside crane has finished lifting w_k in slot 2 and moves right. In case (a), we can assume that the seaside crane waits for an additional p time units because the landside crane will block slot 7 when dropping w_k. □

Additionally, by applying the properties it is also possible to recursively reconstruct concrete drive-

ways of the cranes when beginning with some state of the DP graph that corresponds to a feasible (partial) solution.

4.2.2. Generating the DP Graph

The DP graph is generated recursively. For each state in which at most one crane is active, new states are derived by iterating over all (pairs of) potential job assignments. Identical states are merged. For each state with $c^w = 0$, $list^h = \emptyset$ and with *container* not relating to a seaside container, i.e. a situation in which all seaside containers have reached their target slots, no further states are generated, because it can easily be validated if it represents a feasible solution by checking whether the relevant landside containers are or will be on time.

The generation of the DP graph is illustrated in the following example.

Example 3. Let $S = 6, \sigma_w = 0, \sigma_l = 1, p = 4, n = 4, s_1 = 2, s_2 = 1, s_3 = 5, s_4 = 4, m = 1, a_1 = 4, r_1 = 26, d_1 = 32$. Figure 4 depicts some of the states of the corresponding DP graph. Each state is denoted

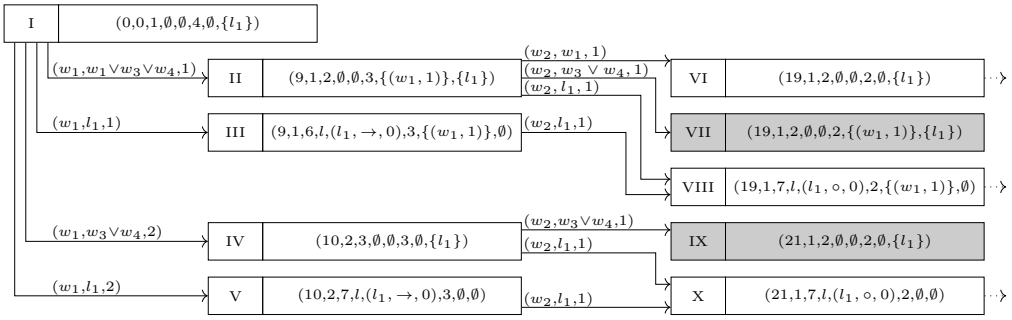


Figure 4: DP graph of Example 3

by a Roman numeral. Each transition is represented by a triple ($w_i \in I, w_k \in I \vee l_j \in J, s \in S$), with w_i indicating the container that is assigned to the seaside crane with destination slot s, and w_k or l_j representing the container that is assigned to the landside crane. s can be any handover slot or the target slot s_i of w_i (if $s_i \neq S + 1$).

Starting from state I, the seaside crane must process w_1 first. Based on Property 3, we can assume that it has to be dropped in slot 1 (handover) or 2 (target slot). The landside crane's job can be any seaside container except w_2 (as this must not be a handover container) or the landside job l_1. If the seaside crane is assigned container w_1 and destination slot 1, there are three transitions with the landside crane processing w_1, w_3 or w_4, each resulting in state II. In Figure 4, these transitions are combined to a single one. Furthermore, state III represents the case where the landside crane is assigned l_1. If the seaside crane's job is to move w_1 to its target slot, state IV corresponds to the landside crane processing w_3 or w_4 and state V relates to the landside crane processing l_1.

Given any of the states II–V, the seaside crane's next job will be moving w_2 to slot 1. States III and V are succeeded by only one state (VIII and X, respectively) because the landside crane is still active and there is no interference. In case of states II and IV, however, the landside crane is available for

processing a new container, such that there are multiple succeeding states generated by iterating over the remaining job assignments, including both seaside and landside containers.

As we will illustrate in Section 4.4, states VII and IX are dominated by state VI and can thus be fathomed.

Obviously, we will have to generate the complete graph in order to derive an optimal solution, which is indicated by the dotted lines in Figure 4. □

4.3. Evaluation of States

We now define a score that indicates how likely a given state in the DP graph results in an improved solution. This score will later be used for a best-first search policy of the DP graph. The objective of PCSP-SL is minimizing $C_{\max}$. Therefore, we could simply evaluate a state $\mathfrak{s} = (t, x_{w,t}, x_{l,t}, \mathit{active}, \mathit{container}, c^w, \mathit{list}^h, \mathit{list}^l)$ by considering time t. However, a state with relatively small t but also very little progress in handling the containers is certainly not promising. Hence, we use the following simple measure of a state's progress:

$$\mathit{prog}(\mathfrak{s}) := \sum_{i=1}^{n-c^w} (y_{i,t} + 2p) - |\mathit{list}^h|2p - \begin{cases} (p+p') & \text{if } \mathit{container} = (w_k \in I, \uparrow, p'), \\ p & \text{if } \mathit{container} = (w_k \in I, \leftarrow \vee \rightarrow \vee \circ, p'), \\ p' & \text{if } \mathit{container} = (w_k \in I, \downarrow, p'), \\ 0 & \text{else.} \end{cases}$$

$\mathit{prog}(\mathfrak{s})$ approximately evaluates the portion of the total seaside workload that has already been completed. Note, however, that the progress of a state $\mathfrak{s}$ may be overrated when using $\mathit{prog}(\mathfrak{s})$. This is due to the fact that there exist instances of PCSP-S in which the landside crane moves left while being loaded with a seaside container in every optimal solution (see Jaehn & Kress, 2018).

Example 3 (continued). In the example of Figure 4, we have $\mathit{prog}(I) = 0$, $\mathit{prog}(II) = \mathit{prog}(III) = 1$, $\mathit{prog}(IV) = \mathit{prog}(V) = 10$, etc.

4.4. Dominance Relations Among States

In order to ensure a reasonable number of states in the DP graph, our DP algorithm checks dominance relations among states.

Example 3 (continued). Consider the example of Figure 4. When comparing the states VI and VII, we find that all properties but the list of handover containers that are waiting to be processed by the landside crane are identical. It is therefore obvious that state VII is dominated and can be fathomed. Similarly, when comparing states VI and IX, the properties are identical except for the time instant so that state IX is dominated.

Consider two distinct states $\mathfrak{s} = (t, x_{w,t}, x_{l,t}, \mathit{active}, \mathit{container}, c^w, \mathit{list}^h, \mathit{list}^l)$ and $\mathfrak{s}' = (t', x'_{w,t'}, x'_{l,t'}, \mathit{active}', \mathit{container}', c^{w'}, \mathit{list}^{h'}, \mathit{list}^{l'}) \neq \mathfrak{s}$ of the current DP graph during the runtime of the

DP procedure, and denote the variables introduced in Section 2 in state $\mathfrak{s}'$ by using an additional prime. Furthermore, given any state $\mathfrak{s}^* = (\dots, \mathit{container}^*, \dots)$, define

$$\bar{p}(\mathfrak{s}^*) := \begin{cases} (p + p') & \text{if } \mathit{container}^* = (w_k \in I \vee l_k \in J, \uparrow, p'), \\ p & \text{if } \mathit{container}^* = (w_k \in I \vee l_k \in J, \leftarrow \vee \rightarrow \vee \circ, p'), \\ p' & \text{if } \mathit{container}^* = (w_k \in I \vee l_k \in J, \downarrow, p'), \\ 0 & \text{else,} \end{cases}$$

which denotes the remaining time needed for lifting and dropping the container that is related to $\mathit{container}^*$ (if existent). We will now define five conditions, denoted by C1 to C5, with respect to $\mathfrak{s}$ and $\mathfrak{s}'$. If all of these conditions apply, we say that $\mathfrak{s}$ *dominates* $\mathfrak{s}'$. Furthermore, $\mathfrak{s}$ and $\mathfrak{s}'$ are then referred to as a *dominating state* and a *dominated state*, respectively. The first three conditions, C1–C3, require the processing of the containers in the dominating state to have progressed at least as much as in the dominated state. Each of the conditions refers to a specific set of containers. C1 relates to seaside containers, C2 takes account of landside containers, and C3 relates to handover containers:

C1: $c^w \leq c^{w'}$: All seaside containers the processing of which has been started by the seaside crane in $\mathfrak{s}'$ have also started to be processed by the seaside crane in $\mathfrak{s}$.

C2: $\mathit{list}^l \subseteq \mathit{list}^{l'}$: All landside containers the processing of which has not yet been started in $\mathfrak{s}$ have also not yet been started to be processed in $\mathfrak{s}'$.

C3: All seaside containers $w_i \in I$ that have been dropped off by the seaside crane at $y_{i,t} < s_i$ in $\mathfrak{s}$, i.e. that are included in list^h, have also been dropped off for handover in $\mathfrak{s}'$ at $y_{i,t}$ or further to the left.

C4 is intended to avoid lengthy case differentiations that are necessary if the seaside crane is currently dropping a container in one of the considered states.

C4: If $\mathit{active} = w$ then $\mathit{container} \neq (\dots, \downarrow, \dots)$ and if $\mathit{active}' = w$ then $\mathit{container}' \neq (\dots, \downarrow, \dots)$: The seaside crane does not currently drop a container in $\mathfrak{s}$ or in $\mathfrak{s}'$.

Finally, C5 takes account of the time that has elapsed in the states. Most important, it makes sure that the cranes in the dominating state are able to reposition within the time difference $t' - t$ such that they are located in favorable slots when compared to the positioning of the cranes in the dominated state. This, of course, depends on the currently active cranes in the states, so that there are quite a few cases to consider:

C5: One of the following is true:

1. $\mathit{active}' = \emptyset$, and
   (a) $\mathit{active} = \emptyset$, $t \leq t' - |x_{l,t} - x'_{l,t'}|$, $t \leq t' - (x_{w,t} - x'_{w,t'})$, and
      i. $t < t' - (x_{w,t} - x'_{w,t'})$, or

ii. $\sum_{w_i \in list^{h'}} y_{i,t} > \sum_{w_i \in list^{h'}} y'_{i,t'}$, or

iii. $c^w < c^{w'}$, or

iv. $list^l \subsetneq list^{l'}$.

(b) $active = w$, $s_{n-c^w} \neq S+1$, $t \leq t' - |x_{l,t} - x'_{l,t'}|$, and

i. $c^w < c^{w'}$, and $t \leq t' - (s_{n-c^w} - x_{w,t} + \bar{p}(\mathfrak{s})) - (s_{n-c^w} - x'_{w,t'})$, or

ii. $c^w = c^{w'}$, $x_{w,t} \geq y'_{n-c^w,t'}$, and $t < t' - \bar{p}(\mathfrak{s}) - (x_{w,t} - x'_{w,t'})$, or

iii. $c^w = c^{w'}$, $x_{w,t} < y'_{n-c^w,t'}$, and $t < t' - (y'_{n-c^w,t'} - x_{w,t} + \bar{p}(\mathfrak{s})) - (y'_{n-c^w,t'} - x'_{w,t'})$.

(c) $active = l$ with the landside crane processing a landside container $l_j \in list^{l'}$, and

i. $t \leq t' - (x_{w,t} - x'_{w,t'})$, and

ii. $container = (l_j, \uparrow \vee \rightarrow \vee \downarrow, p')$, and

iii. $r_j \leq t + S + 1 - x_{l,t} + \bar{p}(\mathfrak{s})$, and

iv. $t \leq t' - (2(S+1) - x_{l,t} - x'_{l,t'} + \bar{p}(\mathfrak{s}))$.

(d) $active = l$ with the landside crane processing a seaside container w_j that is included in $list^{h'}$, and

i. $t \leq t' - (x_{w,t} - x'_{w,t'})$, and

ii. $t \leq t' - (|s_j - x_{l,t}| + |s_j - x'_{l,t'}| + \bar{p}(\mathfrak{s}))$.

2. $active' = w$, and

(a) $active = \emptyset$, and

i. $t \leq t' - |x_{l,t} - x'_{l,t'}|$, and

ii. $t < t' + \bar{p}(\mathfrak{s}') - (x_{w,t} - x'_{w,t'})$.

(b) $active = w$, $t \leq t' - |x_{l,t} - x'_{l,t'}|$, and

i. $c^w = c^{w'}$, and $t < t' - |x'_{w,t'} - x_{w,t}| - (\bar{p}(\mathfrak{s}) - \bar{p}(\mathfrak{s}'))$, or

ii. $c^w < c^{w'}$, $s_{n-c^w} \neq S+1$, and $t \leq t' + \bar{p}(\mathfrak{s}') - (s_{n-c^w} - x_{w,t} + \bar{p}(\mathfrak{s})) - (s_{n-c^w} - x'_{w,t'})$.

(c) $active = l$, and

i. $t < t' + \bar{p}(\mathfrak{s}') - (x_{w,t} - x'_{w,t'})$, and

ii. condition 1. c) ii.–iv. or 1. d) ii.

3. $active' = l$, and

(a) $active = \emptyset$, $t \leq t' - |x_{l,t} - x'_{l,t'}|$, and $t \leq t' - (x_{w,t} - x'_{w,t'})$.

(b) $active = w$, and condition 1. b).

(c) $active = l$, and

i. $t < t' - (x_{w,t} - x'_{w,t'})$, and

ii. the cranes are processing the same container, and

iii. $container = (w_i \in I, \dots)$, and $t \leq t' - |x_{l,t} - x'_{l,t'}| - (\bar{p}(\mathfrak{s}) - \bar{p}(\mathfrak{s}'))$, or $container = (l_j \in J, \dots)$, and $t \leq t' + (x_{l,t} - x'_{l,t'}) - (\bar{p}(\mathfrak{s}) - \bar{p}(\mathfrak{s}'))$.

Whenever there exists a state $\mathfrak{s}$ that dominates another state $\mathfrak{s}'$ within the DP graph, the latter state can be fathomed. However, the DP graph can get very large so that the pairwise comparison of a newly generated state with all existing states can be very time consuming. Hence, we store a list $best(t)$ of the most promising states concerning $prog(\mathfrak{s})/t$ for each potential time instant t and restrict the algorithm to check dominance based on these lists only.

Note that, if $\mathfrak{s}$ dominates one of its direct or indirect successors $\mathfrak{s}'$ in the current DP graph, then there will always exists another successor $\hat{\mathfrak{s}} \neq \mathfrak{s}'$ of $\mathfrak{s}$ that represents the constellation that caused the dominance, so that the DP procedure does not lose its ability to find optimal solutions. The same holds for the case of mutual exclusion of paths of the DP graph due to fathoming states in accordance with the above dominance criteria.

4.5. The DP Algorithm

Our DP procedure makes use of the bucket brigade heuristics as well as the lower bounds summarized in Sections 2.1 and 2.2. Given a state $\mathfrak{s}$, we denote the corresponding lower bound determined by Algorithm 1 (in its adapted version for partial solutions) by $LB^*(\mathfrak{s})$. Additionally, our DP algorithm uses a predetermined search policy for traversing the states of the DP graph. A policy $\mathfrak{P}$ defines two methods. $push(\mathfrak{P}, \mathfrak{S})$ adds a set $\mathfrak{S}$ of states to the policy. $pop(\mathfrak{P})$ retrieves the next state to be processed or an empty set if no more states are available. We have implemented a total of five search policies. On the one hand, we use the traditional policies *depth-first search* (DFS) and *breadth-first search* (BFS). On the other hand, we use a problem specific best-first search policy that we refer to as *prog search* (PROG). It is based on sorting the states $\mathfrak{s}$ in non-increasing order of their values $prog(\mathfrak{s})/t$. $pop(PROG)$ always returns a state $\mathfrak{s}$ with largest value $prog(\mathfrak{s})/t$ among all states that must still be examined by the algorithm. Finally, we combine PROG with DFS and BFS (to DFS PROG and BFS PROG) by sorting the set $\mathfrak{S}$ of states in analogy to the PROG policy, before calling $push(\mathfrak{P}, \mathfrak{S})$ as in DFS and BFS. Every search policy must provide a suitable data structure in the background. In case of DFS, for example, the policy can be implemented as a last in first out (LIFO) stack, while a queue is suitable for implementing BFS. When implementing PROG, one can use a sorted list.

We summarize the previous deliberations in the following algorithm.

Algorithm 2 (Bounded Dynamic Programming)

0. Initialize: Initialize an upper bound $UB := \infty$ and try to improve it using the bucket brigade heuristics SB and BB. Let $\mathfrak{s} = (0, \sigma_w, \sigma_l, \emptyset, \emptyset, n, \emptyset, J)$ be the initial state and $push(\mathfrak{P}, \{\mathfrak{s}\})$. Initialize the DP graph $G = (V := \{\mathfrak{s}\}, E := \emptyset)$ with vertex set V (states) and edge set E (transitions). Furthermore, initialize empty lists $best(t)$, $t = 1, 2, \ldots$, to store promising states.

1. Pop from search policy: Pop the next state $\mathfrak{s} := pop(\mathfrak{P})$. If $\mathfrak{s} = \emptyset$, then stop. If $LB^*(\mathfrak{s}) \geq UB$, then go to Step 1. Let t be the current time instant of $\mathfrak{s}$ and check if $\mathfrak{s}$ is dominated by some $\mathfrak{s}' \in best(t')$, $\mathfrak{s} \neq \mathfrak{s}'$, for any $t' \leq t$. If this is the case, then go to Step 1.

2. **Determine successors:** Determine the set $\mathfrak{S}$ of potential successors of $\mathfrak{s}$ in the DP graph as described in Section 4.2.
3. **Build DP graph:** For each $\mathfrak{s}' = (t', x'_{w,t'}, x'_{l,t'}, active', container', c^{w'}, list^{h'}, list^{l'}) \in \mathfrak{S}$ do:
   - **3.1 Modify DP graph:** Set boolean flag $vis := false$. If $\mathfrak{s}' \in V$ and $\mathfrak{s}'$ is not marked infeasible, set $E := E \cup \{(\mathfrak{s}, \mathfrak{s}')\}$ and go to Step 3.6. Else, if $\mathfrak{s}' \in V$ and $\mathfrak{s}'$ is marked infeasible, set $vis := true$ and $E := E \cup \{(\mathfrak{s}, \mathfrak{s}')\}$. Else, set $V := V \cup \{\mathfrak{s}'\}$ and $E := E \cup \{(\mathfrak{s}, \mathfrak{s}')\}$.
   - **3.2 Feasibility:** Check if all landside containers that have been dropped in $S+1$ or that are currently handled by the landside crane in $\mathfrak{s}'$ are or will be on time. If this is not the case, go to Step 3.6.
   - **3.3 New best solution:** If $c^{w'} = 0$, $list^{h'} = \emptyset$, and if $container'$ does not relate to a seaside container, then check solution for feasibility by checking if an additional landside container $l_{j^{next}}$ (see Section 2) can be dropped off on time. If this is the case, potentially update UB and go to Step 3.6. If infeasibility is detected, go to Step 3.6. If $vis = true$, go to Step 3.7.
   - **3.4 Bounding:** If $LB^*(\mathfrak{s}') \geq UB$, go to Step 3.6.
   - **3.5 Update best states:** If $best(t') = \emptyset$ or $prog(\mathfrak{s}')/t' \geq prog(\hat{\mathfrak{s}})/t'$ for some $\hat{\mathfrak{s}} \in best(t')$, then insert $\mathfrak{s}'$ into $best(t')$. If $|best(t')| > 5$, eliminate a state with worst progress. Go to Step 3.7.
   - **3.6 Remove successor:** Set $\mathfrak{S} := \mathfrak{S} \setminus \{\mathfrak{s}'\}$ ($\mathfrak{s}'$ is fathomed or was already existing).
   - **3.7 Next successor:** Continue Step 3.
4. **Push to search policy:** Add the remaining states in $\mathfrak{S}$ to the search policy for further processing, i.e. $push(\mathfrak{P}, \mathfrak{S})$, and go to Step 1.

Note that in Steps 3.2 and 3.6, Algorithm 2 might fathom a state because a landside container cannot be dropped in slot $S+1$ on time, even though this container may actually not have to be processed in a succeeding feasible solution (see Section 1.1). In this case, however, the process of generating states as outlined in Section 4.2 prevents the algorithm from missing a related optimal solution.

4.6. Beam Search

Due to the flexibility of Algorithm 2 with respect to the incorporation of different search policies, it is easily possible to transform the exact DP algorithm into a beam search heuristic. In general, BS uses a graph representation of the solution process and applies breadth-first search with a filtering process to only expand the most promising nodes of the graph. It was first used by Lowerre (1976). A review is provided by Sabuncuoğlu et al. (2008).

Our BS procedure for PCSP-SL combines the BFS policy in Algorithm 2 with a simple filtering method that selects β states $\mathfrak{s}$ with largest ratios $prog(\mathfrak{s})/t$ of the current level of the DP graph for further consideration. All remaining states of a level are excluded from further consideration in the DP

algorithm. The parameter β is usually referred to as the *beam width*. The *level* of a state $\mathfrak{s}$ is defined as the length (in terms of the number of edges in the DP graph) of the shortest path from the initial state to $\mathfrak{s}$ when the state is first inserted into the DP graph in Step 0 or Step 3.2 of Algorithm 2. Details of our BS implementation are presented in Appendix D.

5. Computational Study

In order to assess the performance and practical applicability of our DP algorithm and the corresponding BS heuristic, we performed a comprehensive computational study that was driven by five research questions, Q1–Q5, which we will evaluate in this section. All algorithms were implemented in C++ (Microsoft Visual Studio Professional 2013). The experiments were executed on a PC with an Intel® Core™ i7-4770 CPU running at 3.4 GHz and 16 GB of RAM under a 64-bit version of Windows 8.

Q1 and Q2 are concerned with the performance of the DP algorithm for small instances. Based on a mixed-integer program that we extend to include landside containers in Appendix A, Briskorn et al. (2016) find that "only the small instances [of PCSP-S] ($|I| = 5$, $S = 5$, and $p = 1$) could be solved to optimality [by CPLEX 12.5 on a similar PC] within the time limit of 60 min." The research questions are as follows:

Q1: Can the DP procedure solve small instances of PSCP-SL to optimality in reasonable time?

Q2: Is the DP algorithm able to quickly solve instances of PCSP-S to optimality that are larger than the ones solved by Briskorn et al. (2016) by using CPLEX?

Positive answers to Q1 and Q2 justify the existence of our DP procedure and are a prerequisite for answering Q3 that seeks to investigate the practical applicability of the DP algorithm.

Q3: Is it possible to obtain optimal solutions or at least improve the results of the bucket brigade heuristics for larger instance sizes of PCSP-S and PCSP-SL in adequate time?

Due to the computational complexity of PCSP-SL, we cannot expect the DP procedure to be able to solve instances with hundreds of containers and large storage blocks to optimality. Hence, "larger instance sizes" in the context of Q3 refers to real-world yard settings regarding S and p, but only few containers. Such problem settings can, for instance, be realistic when considering an online setting of PCSP-SL with the data for only a few containers being available when executing the algorithm (see, e.g. Dorndorf & Schneider, 2010, for a similar online setting). Briskorn et al. (2016) provide data from the Europe Container Terminals (ECT) at the port of Rotterdam for constructing real-world instances of PCSP-SL. Here, a block is typically about 40 TEU long, which corresponds to $S = 40$ (see also Saanen & Valkengoed, 2005), and cranes move along the block at a speed of around 3 meters per second so

that a time interval of PCSP-SL corresponds to roughly two seconds of real time. Furthermore, the authors show that $p = 20$ is a realistic assumption.

Finally, Q4 and Q5 aim at analyzing the heuristic BS approach.

Q4: How does BS perform with respect to solution quality and runtime when using a cold start, i.e. when omitting the calculation of an upper bound with the bucket brigade heuristics in Step 0 of Algorithm 2?

Q5: Is it possible to (quickly) improve the results obtained by the bucket brigade heuristics using BS?

5.1. Instance Generation

As Q1 is of theoretical nature, we constructed relatively small instances in order to answer this research question. This holds for both, the size of the storage yard and the number of containers when compared to the real-world parameters. Table 4 depicts the parameter values that our testbed regarding Q1 is based upon. For each combination of the parameter values of Table 4, we generated

Table 4: Parameters of PCSP-SL test instances for Q1 and Q4

fixed	variable				
σ_w	S	p	σ_l	n	m
0	5, 7	1, 4, 6	$\lceil \frac{S}{2} \rceil$	5, 8, 10	1, 2, 4

3 test instances, which results in a total of 162 test instances. The same instances are used for our analysis of Q4.

The parameter values of the test instances for Q2 and Q3 are presented in Table 5. We generated 10

Table 5: Parameters of PCSP-S and PCSP-SL test instances for Q2 and Q3

fixed	variable				
σ_w	S	p	σ_l	n	m (PCSP-SL only)
0	5, 10, 15, 20, 40	1, 5, 10, 15, 20	$\lfloor \frac{S}{2} \rfloor$	5, 10, 15, 20	$\lfloor \frac{n}{3} \rfloor$

test instances for each combination of the parameter values, i.e. a total of 1,000 instances for PCSP-S and another 1,000 instances for PCSP-SL. As mentioned above, some of these instances mimic real-world yard settings regarding S and p with only few containers, which can be a realistic situation in online settings of PSCP-S or PCSP-SL.

The parameters presented in Table 6 were used to generate instances for Q5. We generated 20 in-

Table 6: Parameters of PCSP-SL test instances for Q5

fixed	variable				
σ_w	S	p	σ_l	n	m
0	20, 40	$\frac{S}{2}$	$\frac{S}{2}$	5, 10, 15	$\lfloor \frac{n}{3} \rfloor$

stances for each parameter combination, i.e. 120 instances in total. Again, these instances can represent realistic situations in online settings of PCSP-SL.

The remaining parameters of the test instances were determined in analogy to Jaehn & Kress (2018), i.e. they were randomly drawn from uniform distributions on the intervals depicted in Table 7, where $\bar{C}_{sea} = 2pn + Sn - \frac{S}{2}$ is the expected seaside makespan when assuming that there exist no landside containers and that preemption of jobs is not allowed. The parameters δ_r and δ_d were applied to

Table 7: Intervals of uniform distributions for generating the container data

$s_i,\, w_i \in I$	$a_j,\, l_j \in J$	$r_j,\, l_j \in J$	$d_j,\, l_j \in J$
$[1, S+1]$	$[1, S]$	$[0, \lfloor \delta_r \cdot \bar{C}_{sea} \rfloor]$	$[r_j, \lfloor \delta_d \cdot \delta_r \cdot \bar{C}_{sea} \rfloor]$

generate time windows for the landside jobs. As shown by Jaehn & Kress (2018), $\delta_r = 0.7$ and $\delta_d = 1.3$ result in a large share of completed landside jobs in heuristic solutions to instances of PCSP-SL, which is why we have chosen these values for our computational tests.

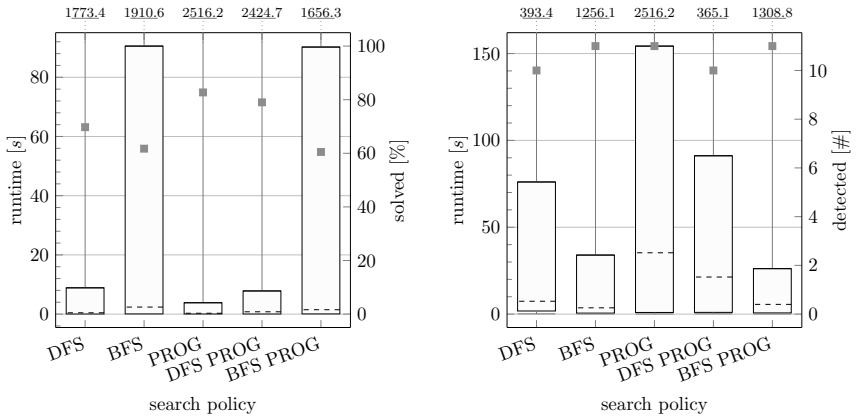
We assume that all data is available at the beginning of the planning horizon throughout this computational study. This is a realistic assumption in light of the fact that we restrict our attention to instances with only a few containers. Within the planning process at seaports, PCSP-SL will, in general, be embedded into higher level scheduling problems that successively provide short seaside container sequences and a few container requests on the landside.

5.2. Results and Evaluation

The results of our computational study are presented in detail in the following sections.

5.2.1. Research Question Q1 - Small Instances

In order to assess the quality of our exact approach for the small test instances, we evaluated its performance with the search policies DFS (depth-first search), BFS (breadth-first search), PROG (best-first search based on the progress measure introduced in Section 4.3), as well as the hybrids DFS PROG and BFS PROG, as presented in Section 4.5. We imposed a time limit of 45 minutes for each instance-policy combination. The results are presented in Figure 5. Each boxplot (left ordinate) depicts the



(a) Overall performance (b) No feasible solution

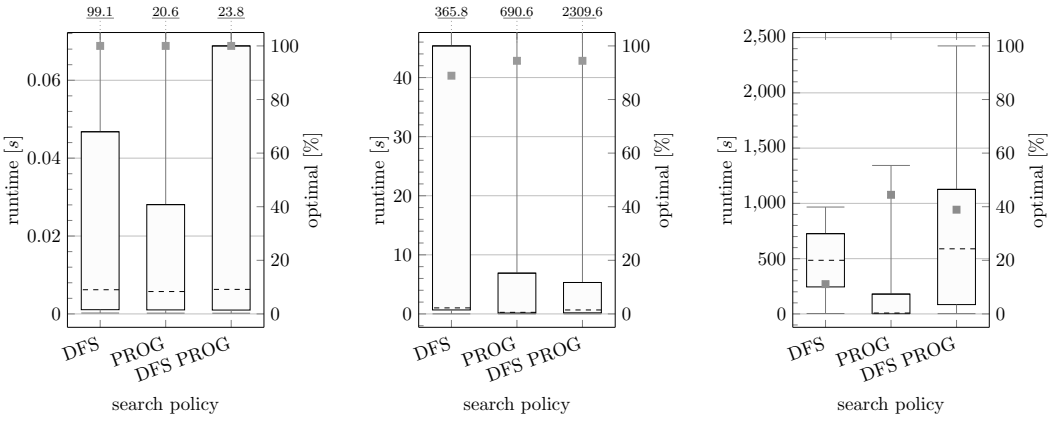
Figure 5: Q1 results, all instances, time limit: 2,700 s

runtime performance of a given search policy over all instances that were solved to optimality (Figure

5a, including the instances without a feasible solution) and that were proven to not have a feasible solution (Figure 5b). It depicts the first quartile of the runtimes (bottom of the box), the third quartile of the runtimes (top of the box), the median of the runtimes (dotted line within the box), the minimum runtime (bottom whisker), and the maximum runtime (top whisker). As the top whiskers lie outside of the coordinate systems for all boxplots, the maximum runtimes are additionally depicted above the whiskers. Note that the boxplots solely consider instances which are solved within the time limit of 2,700 s, so that the maximum runtimes are smaller than 2,700 s. Furthermore, squares inside the charts (right ordinate) of Figure 5 represent the percentage of the instances that were solved to optimality (Figure 5a) and the number of instances that were proven to not have a feasible solution (Figure 5b).

It can be concluded that, while relatively reliably detecting infeasibility, BFS and BFS PROG solve the least instances to optimality and feature the worst runtime performance of all considered search policies. We will therefore exclude these policies from our further analysis. Furthermore, with respect to the overall performance and the ability to detect instances without a feasible solution, PROG clearly outperforms DFS and DFS PROG. When using PROG, the DP solves more than 80% of the instances to optimality, with 75% of these instances being solved in less than four seconds. The relatively large maximum runtime of PROG in comparison to DFS and DFS PROG in Figure 5b is induced by the additional instance that is proven to not have a feasible solution.

In analogy to Figure 5a, Figure 6 presents some more details by focusing on three subsets of instances with $n = 5$ and $m = 1$ (Figure 6a), $n = 8$ and $m = 2$ (Figure 6b), and $n = 10$ and $m = 4$ (Figure 6c). As can be seen from these figures, our DP performs well for the Q1 test instances as long as the number



(a) $n = 5$, $m = 1$ (b) $n = 8$, $m = 2$ (c) $n = 10$, $m = 4$

Figure 6: Q1 results, grouped by instance size, time limit: 2,700 s

of containers remains sufficiently small. Again, PROG can be identified to be the search strategy that performs best.

Summing up, the PROG policy seems most appropriate amongst the strategies presented in this article. Furthermore, we can conclude that dynamic programming in combination with bounding techniques and dominance relations is well suited for solving small instances of PCSP-SL in a reasonable

amount of time in the context of the problem's computational complexity. Q1 can therefore be positively answered. The question of whether or not our DP also outperforms CPLEX is answered in the next section.

5.2.2. Research Questions Q2 and Q3 - Large Instances

For research questions Q2 and Q3, we restricted ourselves to using the PROG policy. The time limit was set to 10 minutes, as Q2 and Q3 aim to analyze the potential of using the proposed DP in real-world settings. Below, we will additionally present results for a larger time limit of 45 minutes. The results are given in Tables 8 (PCSP-S instances) and 9 (PCSP-SL instances). For each set of test instances with given S, p, and n, the tables depict the percentage of instances that were solved to optimality (column *opt.*) within the time limit, the average computational time needed to compute the optimal solutions (column t_{avg}), and the percentage of instances for which the solution determined by the bucket brigade heuristics in Step 0 of Algorithm 2 was improved (column *impr.*).

Table 8: Results for PCSP-S, time limit: 600 s

S	p	$n=5$			$n=10$			$n=15$			$n=20$		
		opt. [%]	t_{avg} [s]	impr. [%]	opt. [%]	t_{avg} [s]	impr. [%]	opt. [%]	t_{avg} [s]	impr. [%]	opt. [%]	t_{avg} [s]	impr. [%]
5	1	100	0.001	60	100	0.016	100	100	17.258	100	90	28.797	100
	5	100	0.002	70	100	0.086	100	100	56.227	100	60	132.24	100
	10	100	0.002	90	100	0.244	100	90	3.775	100	0	-	100
	15	100	0.001	70	100	0.327	100	90	72.078	100	10	4.19	100
	20	100	0.001	70	100	0.087	100	100	19.24	100	10	0.002	100
10	1	100	0.008	80	80	67.428	100	10	36.049	80	0	-	50
	5	100	0.038	90	90	105.211	100	50	140.829	100	10	4.460	90
	10	100	0.007	100	90	3.261	100	30	159.61	100	0	-	100
	15	100	0.075	80	80	27.714	100	30	49.005	100	0	-	90
	20	100	0.022	100	80	63.962	100	0	-	100	0	-	90
15	1	100	1.381	60	10	185.064	70	0	-	20	0	-	20
	5	100	15.46	80	40	4.44	100	10	192.116	80	0	-	60
	10	100	0.337	100	70	198.443	100	10	148.414	90	0	-	70
	15	100	2.461	90	50	99.606	100	0	-	80	0	-	80
	20	100	0.233	90	70	123.728	100	0	-	90	0	-	90
20	1	100	8.038	100	0	-	60	0	-	60	0	-	20
	5	90	4.997	100	10	229.31	90	0	-	50	0	-	30
	10	100	1.482	90	50	192.785	100	0	-	90	0	-	40
	15	100	9.467	80	40	189.513	100	0	-	100	0	-	60
	20	100	5.072	100	40	166.812	100	0	-	90	0	-	70
40	1	30	77.879	80	0	-	50	0	-	0	0	-	10
	5	20	13.147	90	0	-	30	0	-	10	0	-	0
	10	70	82.872	100	0	-	60	0	-	10	0	-	20
	15	60	37.08	100	0	-	40	0	-	20	0	-	0
	20	60	33.734	100	0	-	40	0	-	20	0	-	0

It turns out that our DP procedure performs significantly better than CPLEX, which has shown to only be able to solve very small instances ($n = 5$, $S = 5$, $p = 1$) of PCSP-S to optimality within a time limit of 60 minutes by Briskorn et al. (2016). Basically, the authors make use of a restricted version of the mixed-integer program for PCSP-SL presented in Appendix A, which drops all variables and constraints needed to model the processing of landside containers. In contrast to these results,

Table 9: Results for PCSP-SL, time limit: 600 s

S	p	$n=5$			$n=10$			$n=15$			$n=20$		
		opt. [%]	t_{avg} [s]	impr. [%]	opt. [%]	t_{avg} [s]	impr. [%]	opt. [%]	t_{avg} [s]	impr. [%]	opt. [%]	t_{avg} [s]	impr. [%]
5	1	100	0.003	90	90	33.56	90	0	-	50	0	-	10
	5	100	0.007	100	80	27.786	80	50	176.371	80	0	-	100
	10	100	0.003	90	90	21.38	100	10	3.156	90	0	-	100
	15	100	0.002	100	100	19.72	100	20	176.157	100	0	-	100
	20	100	0.002	80	90	26.86	90	10	77.286	80	0	-	100
10	1	100	0.902	90	0	-	20	0	-	30	0	-	0
	5	90	0.019	90	50	111.76	100	10	198.728	80	0	-	60
	10	100	0.031	80	50	8.679	80	10	502.202	100	0	-	100
	15	100	0.098	80	60	124.773	100	0	-	100	0	-	90
	20	100	0.047	90	80	36	90	0	-	80	0	-	90
15	1	90	0.74	100	0	-	20	0	-	0	0	-	0
	5	100	2.414	90	20	20.406	90	0	-	50	0	-	50
	10	100	2.495	90	20	52.097	90	0	-	90	0	-	70
	15	100	1.94	90	20	22.83	100	0	-	90	0	-	80
	20	100	0.047	100	30	200.192	100	0	-	90	0	-	100
20	1	40	9.24	100	0	-	20	0	-	0	0	-	10
	5	90	64.928	100	0	-	70	0	-	20	0	-	10
	10	90	9.853	100	0	-	80	0	-	90	0	-	30
	15	90	2.165	90	0	-	40	0	-	80	0	-	50
	20	100	0.578	100	0	-	100	0	-	80	0	-	50
40	1	0	-	70	0	-	30	0	-	0	0	-	20
	5	20	6.447	90	0	-	10	0	-	10	0	-	0
	10	0	-	80	0	-	30	0	-	0	0	-	0
	15	30	159.474	80	0	-	40	0	-	40	0	-	20
	20	20	68.245	80	0	-	20	0	-	10	0	-	10

we can now quickly solve instances with up to 40 slots, realistic lift and drop times, and few landside containers to optimality, both for PCSP-S and PCSP-SL. As to be expected, the incorporation of landside containers results in larger running times and a smaller share of instances than can be solved to optimality. However, there exist instances of PCSP-SL with $S = 10$ and $n = 15$ that our DP solves to optimality. This clearly allows for positively answering Q2.

Additionally, we observe that the DP algorithm is capable of improving solutions provided by the bucket brigade heuristics for real-world yard settings with few containers. Hence, we can positively answer Q3 as well. The computational tests show that, given a time limit of a few minutes, it is rewarding to apply our DP algorithm to try to improve solutions provided by the bucket brigade heuristics.

Table 10 illustrates the effect of a larger time limit of 45 minutes on the performance of the DP with respect to the percentage of instances that can be solved to optimality. We called the DP with the increased time limit on all of the above instances that were not solved to optimality within the 10 minutes limit. The table depicts the percentage of these calls that returned an optimal solution. A dash marks the settings where no call of the DP was necessary because all instances were solved to optimality within the small time limit. The results support our above findings for research question Q2, as quite a few additional instances were solved to optimality within the increased time limit in case

Table 10: Effect of increasing the time limit to 2,700 s

S	p	$n = 5$		$n = 10$		$n = 15$		$n = 20$	
		PCSP-S	PCSP-SL	PCSP-S	PCSP-SL	PCSP-S	PCSP-SL	PCSP-S	PCSP-SL
5	1	-	-	-	0	-	20	100	0
	5	-	-	-	0	-	20	25	0
	10	-	-	-	100	100	11.11	10	0
	15	-	-	-	-	100	12.50	11.11	0
	20	-	-	-	0	-	33.33	22.22	0
10	1	-	-	50	0	22.22	0	0	0
	5	-	100	0	0	0	0	0	0
	10	-	-	100	20	14.29	0	0	0
	15	-	-	50	100	0	10	0	0
	20	-	-	0	0	20	0	0	0
15	1	-	100	0	0	0	0	0	0
	5	-	-	50	0	0	0	0	0
	10	-	-	33.33	0	0	0	0	0
	15	-	-	40	50	0	0	0	0
	20	-	-	33.33	28.57	0	0	0	0
20	1	-	66.67	10	0	0	0	0	0
	5	0	100	22.22	0	0	0	10	0
	10	-	0	20	0	0	0	0	0
	15	-	0	16.67	0	20	0	0	0
	20	-	-	16.67	10	0	0	0	0
40	1	42.86	0	0	0	0	0	0	0
	5	37.5	12.5	0	0	0	0	0	0
	10	0	10	0	0	0	0	0	0
	15	25	28.57	0	0	0	0	0	0
	20	50	37.5	10	0	0	0	0	0

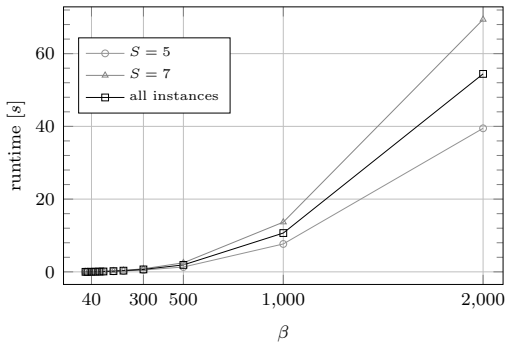
of both PCSP-S and PCSP-SL.

5.2.3. Research Questions Q4 and Q5 - Beam Search

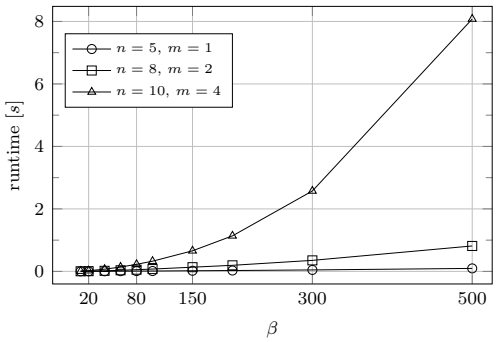
For our computational tests regarding the heuristic BS approach, we terminated Algorithm 2 (in addition to the regular termination criteria) upon finding (in case of Q4) or improving (in case of Q5) a feasible solution for a test instance or when a time limit of 15 minutes was reached.

Regarding the Q4 test instances, we executed the BS algorithm with multiple beam widths $\beta \in \{10, 20, 40, 60, 80, 100, 150, 200, 300, 500, 1000, 2000\}$ for each test instance. The results are presented in Figure 7.

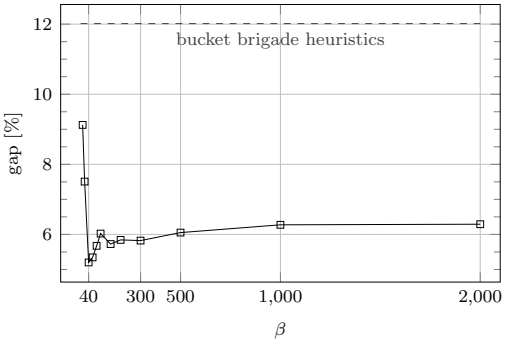
Figures 7a and 7b depict the average runtime behaviour of the BS approach over the beam width. Figure 7a shows that the average runtime over all instances ranges from only a few milliseconds for small beam widths to about 55 seconds for a beam width of 2,000. The figure additionally includes the corresponding curves for the instance subsets with five or seven storage slots, respectively. As to be expected, a larger storage area induces the need for more computational effort. The same is true when increasing the total number of containers, which can be seen in Figure 7b, where runtime plots of three subsets of the test instances, i.e. instance sets with $n = 5$ and $m = 1$, $n = 8$ and $m = 2$, and $n = 10$ and $m = 4$, are presented. As a consequence, β must be chosen carefully in order to be able to determine a feasible solution within a reasonable amount of time when facing real-world instances of



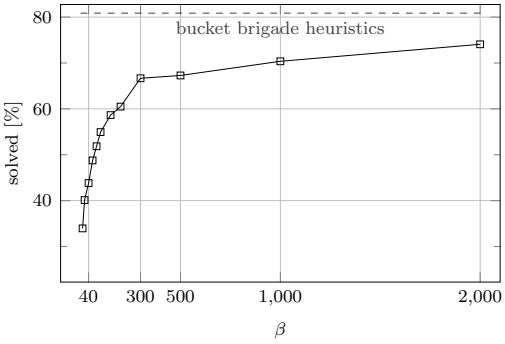
(a) Average runtime



(b) Average runtime: effect of container number



(c) Average gap



(d) Avg. # of inst. for which a feas. sol. was found

Figure 7: Beam search performance for Q4 instances, time limit: 900 s

PCSP-SL.

Figures 7c and 7d compare the quality of the solutions determined by the BS approach to the quality of the solutions determined by the bucket brigade heuristics. Figure 7c is based on all instances I, for which a feasible solution with objective function value $C_{max}^{heur}(I)$ was found with the respective heuristic *heur*, and for which the optimal objective function value $C_{max}^{opt}(I)$ is known based on our tests in Section 5.2.1. It plots average values of the optimality gaps, $gap^{heur}(I) = (C_{max}^{opt}(I) - C_{max}^{heur}(I))/C_{max}^{opt}(I)$, over the beam width as well as the corresponding average optimality gap related to the bucket brigade heuristics. Similarly, Figure 7d depicts the percentage of instances for which a feasible solution was found by the heuristic approaches. We conclude that, while the BS approach is less reliable than the bucket brigade heuristics with respect to finding feasible solutions for small instances of PCSP-SL, it outperforms the bucket brigade algorithms in terms of solution quality. As the BS approach under consideration for Q4 uses a cold start, it is reasonable to expect that a warm start that applies the bounds determined by the bucket brigade heuristics can further improve the runtime and quality performance. This is subject of research question Q5.

Based on our above deliberations on Q4, we conclude that it is unlikely that the BS approach will improve a feasible solution determined by the bucket brigade algorithms in adequate time for real-world problem instances with hundreds of containers, which is why the Q5 test instances presented in Section 5.1 focus on real-world yard settings with few containers. The corresponding results of our computational tests are presented in Table 11. As a reasonable compromise between runtime and

solution quality, we decided to set $\beta = 750$ for instances with $S = 20$ and $\beta = 500$ for instances with $S = 40$, based on the results presented in Figure 7. Table 11 depicts the percentage of instances

Table 11: Beam search performance for Q5 instances, time limit: 900 s

S	β	feas. sol. [%]	improvement			no improvement	
			count [%]	t_{avg} [s]	avg. impr. [%]	count [%]	t_{avg} [s]
20	750	96.67	38.33	8.71	7.94	61.67	302.12
40	500	93.33	23.33	12.56	10.81	76.67	376.09

for which the warm start variant of our BS approach returned a feasible solution (column *feas. sol.*) for the instance sets with 20 and 40 storage slots, respectively. The table then differentiates between the instances for which the bucket brigade solution was improved and the instances for which no improvement was found. It provides details on the relative amount of instances (column *count*), the average computational time (column t_{avg}), and the average improvement of the makespan (column *avg. impr.*). We observe that a significant improvement of the objective function value was computed in reasonable time for a substantial amount of instances. We therefore conclude by positively answering Q5 so that the warm start variant of our BS approach seems well suited for real-world online settings of PCSP-SL.

6. Conclusion

In this article, we have presented an exact dynamic programming procedure for the preemptive crane scheduling problem with a given unloading sequence and additional landside jobs (PCSP-SL) that has recently been introduced and analyzed in Briskorn et al. (2016) and Jaehn & Kress (2018). The procedure makes use of bounding techniques and applies various dominance properties of optimal solutions that we have introduced.

In extensive computational tests, we have shown that, among the five search policies that we have introduced in this article, a problem specific best-first search policy that is based on a simple measure of a state's progress performs best. When using this strategy, the DP has shown to perform very well for small instances with few storage slots as long as the number of containers remains sufficiently small. Furthermore, the DP has shown to be able to quickly improve solutions determined by the bucket brigade heuristics presented in Jaehn & Kress (2018) and to clearly outperform CPLEX in terms of its ability to compute optimal solutions in reasonable time for small to medium sized instances with real-world yard settings and few containers. This allows the algorithm to be applied in real-world online settings of PCSP-SL, where container data is revealed incrementally.

In addition to our DP, we have introduced a beam search heuristic for PCSP-SL. In line with the results for the exact approach, this heuristic has shown to be capable of improving the makespan of solutions determined by the bucket brigade heuristics in reasonable time when few containers are considered, so that it is a good alternative to the DP in real-world online settings of PCSP-SL.

These findings lead to some future research directions. Even though it is well-accepted that many interacting decisions in seaports are decomposed and treated hierarchically, the positive results obtained in this paper result in the question of whether some of these decisions could be combined. It should thus be evaluated whether the combined consideration of the problem at hand with related problems is reasonable. These related problems include the assignment of containers to blocks, the positioning of the containers in the block, and the sequencing of the seaside containers. Moreover, one could more thoroughly analyze the effect of serving landside containers on a vessel's berthing time. As mentioned before, seaside operations usually have much higher priority than landside actions and obviously, serving landside containers may increase the berthing time of a vessel. As some high priority landside actions are considered to be very valuable, knowledge on their effect on the berthing time allows for decisions on how many landside actions are acceptable and how pricing schemes for these actions should be designed.

Acknowledgements

This work was supported by the German Science Foundation (DFG) through the grant "Scheduling mechanisms for rail mounted gantries with regard to crane interdependencies" (JA 2311/2-1, PE 514/22-1).

References

Bierwirth, C., & Meisel, F. (2010). A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, *202*, 615–627.

Bierwirth, C., & Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, *244*, 675–689.

Boysen, N., Briskorn, D., & Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, *258*, 343–357.

Briskorn, D., Emde, S., & Boysen, N. (2016). Cooperative twin-crane scheduling. *Discrete Applied Mathematics*, *211*, 40–57.

Carlo, H. J., Vis, I. F. A., & Roodbergen, K. J. (2014). Storage yard operations in container terminals: literature overview, trends, and research directions. *European Journal of Operational Research*, *235*, 412–430.

Dorndorf, U., & Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum*, *32*, 617–632.

Ehleiter, A., & Jaehn, F. (2016). Housekeeping: foresightful container repositioning. *International Journal of Production Economics*, *179*, 203–211.

Gharehgozli, A. H., Vernooij, F. G., & Zaerpour, N. (2017). A simulation study of the performance of twin automated stacking cranes at a seaport container terminal. *European Journal of Operational Research*, *261*, 108–128.

Jaehn, F., & Kress, D. (2018). Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete Applied Mathematics*, *242*, 53–68.

Kemme, N. (2012). Effects of storage block layout and automated yard crane systems on the performance of seaport container terminals. *OR Spectrum*, *34*, 563–591.

Kovalyov, M. Y., Pesch, E., & Ryzhikov, A. (2018). A note on scheduling container storage operations of two non-passing stacking cranes. *Networks*, *71*, 271–280.

Lashkari, S., Wu, Y., & Petering, M. E. (2017). Sequencing dual-spreader crane operations: Mathematical formulation and heuristic algorithm. *European Journal of Operational Research*, *262*, 521–534.

Lee, C.-Y., & Song, D.-P. (2017). Ocean container transport in global supply chains: Overview and research opportunities. *Transportation Research Part B: Methodological*, *95*, 442–474.

Lowerre, B. T. (1976). *The HARPY Speech Recognition System*. Ph.D. thesis Carnegie-Mellon University Pittsburgh, Pennsylvania.

Port of Rotterdam (2017). Port facts and figures: throughput. `https://www.portofrotterdam.com/en/the-port/port-facts-and-figures/throughput`. (last accessed: 02/10/2017).

Saanen, Y. A., & Valkengoed, M. V. (2005). Comparison of three automated stacking alternatives by means of simulation. In *Proceedings of the 37th Conference on Winter Simulation* (pp. 1567–1576). ACM.

Sabuncuoğlu, I., Gocgun, Y., & Erel, E. (2008). Backtracking and exchange of information: Methods to enhance a beam search algorithm for assembly line scheduling. *European Journal of Operational Research*, *186*, 915–930.

Speer, U., & Fischer, K. (2017). Scheduling of different automated yard crane systems at container terminals. *Transportation Science*, *51*, 305–324.

Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: a literature update. *OR Spectrum*, *30*, 1–52.

Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operations and operations research – a classification and literature review. *OR Spectrum*, *26*, 3–49.

Appendix A. Mixed-Integer Program

Let T be an upper bound on the number of time slots needed to characterize an optimal solution of an instance of PCSP-SL. The variables $x_{c,t}$, $c \in \{w, l\}$, $t \in \{0, \ldots, T\}$, have already been defined in Section 2. Now, define the following additional binary variables:

$$l_t^I := \begin{cases} 1, & \text{if crane } c = w \text{ starts lifting any container } w_i \in I \text{ at time instant } t \\ 0, & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, T\},$$

$$l_{t,i,s}^I := \begin{cases} 1, & \text{if crane } c = l \text{ starts lifting container } w_i \text{ at} \\ & \text{time instant } t \text{ in slot } s \\ 0, & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, T\}, s \in \{1, \ldots, S\}, w_i \in I,$$

$$l_{t,j}^J := \begin{cases} 1, & \text{if crane } c = l \text{ starts lifting container } l_j \text{ at time instant } t \\ 0, & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, T\}, l_j \in J$$

$$d_{t,i,s}^I := \begin{cases} 1, & \text{if crane } c = w \text{ starts dropping container} \\ & w_i \text{ at time instant } t \text{ in slot } s \\ 0, & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, T\}, s \in \{1, \ldots, S\}, w_i \in I,$$

$$d_{t,i}^I := \begin{cases} 1, & \text{if crane } c = l \text{ starts dropping container } w_i \text{ at time instant } t \\ 0, & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, T\}, w_i \in I,$$

$$d_{t,j}^J := \begin{cases} 1, & \text{if crane } c = l \text{ starts dropping container } l_j \text{ at time instant } t \\ 0, & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, T\}, l_j \in J.$$

Additionally, we make use of a nonnegative variable $C \in \mathbb{R}_0^+$ which represents the makespan, as well as binary variables u_j, v_j and q_j, $l_j \in J$. u_j is set to one if landside container $l_j \in J$ must be processed because its deadline is not greater than C. Similarly, the binary variables v_j, $l_j \in J$, assure that (in case of its existence) an additional landside container with smallest deadline larger than C is processed by the landside crane. The variables q_j, $l_j \in J$, are needed for modelling purposes.

Now, let $n \geq 1$ and assume without loss of generality that the landside jobs of the set J are ordered in non-decreasing order of the deadlines of the respective jobs. Define a parameter λ_j for each landside job $l_j \in J$, which is set to the number of landside jobs with the same deadline as l_j. Define $\lambda_{m+1} = 0$ and $v_j = 0$ for all $j \in \{m+1, \ldots, m+\lambda_m\}$. Then, based on the mixed-integer program for PCSP-S by Briskorn et al. (2016), a mathematical formulation of PCSP-SL is as follows.

$$\min \quad C \tag{A.1}$$

$$\text{s.t.} \quad t \cdot d_{t,n,s_n}^I + p \leq C \qquad \forall t \in \{1, \ldots, T\}, \tag{A.2}$$

$$t \cdot \sum_{w_i \in I} d^I_{t,i} + p \leq C \qquad \forall t \in \{1, \ldots, T\}, \tag{A.3}$$

$$x_{c,0} = \sigma_c \qquad \forall c \in \{w, l\}, \tag{A.4}$$

$$x_{c,t-1} - 1 \leq x_{c,t} \leq x_{c,t-1} + 1 \qquad \forall c \in \{w, l\}, t \in \{0, \ldots, T\}, \tag{A.5}$$

$$x_{w,t} \leq x_{l,t} - 1 \qquad \forall t \in \{0, \ldots, T\}, \tag{A.6}$$

$$x_{c,t} \leq S + 1 \qquad \forall c \in \{w, l\}, t \in \{1, \ldots, T\}, \tag{A.7}$$

$$\sum_{t=0}^{T} \sum_{s=1}^{S} t \cdot d^I_{t,i,s} \leq \sum_{t=0}^{T} \sum_{s=1}^{S} t \cdot d^I_{t,j,s} \qquad \forall w_i, w_j \in I, i < j \tag{A.8}$$

$$x_{w,t'} \leq (1 - l^I_t) \cdot S \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p\} \tag{A.9}$$

$$\begin{aligned}(1 - \sum_{w_i \in I} \sum_{s=1}^{S} d^I_{t,i,s}) \cdot S \\ + \sum_{w_i \in I} \sum_{s=1}^{S} s \cdot d^I_{t,i,s} \geq x_{w,t'} \geq \sum_{w_i \in I} \sum_{s=1}^{S} s \cdot d^I_{t,i,s}\end{aligned} \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p\} \tag{A.10}$$

$$\begin{aligned}(1 - \sum_{w_i \in I} \sum_{s=1}^{S} l^I_{t,i,s}) \cdot (S+1) \\ + \sum_{w_i \in I} \sum_{s=1}^{S} s \cdot l^I_{t,i,s} \geq x_{l,t'} \geq \sum_{w_i \in I} \sum_{s=1}^{S} s \cdot l^I_{t,i,s}\end{aligned} \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p\} \tag{A.11}$$

$$\begin{aligned}(1 - \sum_{w_i \in I} d^I_{t,i}) \cdot (S+1) \\ + \sum_{w_i \in I} s_i \cdot d^I_{t,i} \geq x_{l,t'} \geq \sum_{w_i \in I} s_i \cdot d^I_{t,i}\end{aligned} \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p\} \tag{A.12}$$

$$\begin{aligned}(1 - \sum_{l_i \in J} l^J_{t,j}) \cdot (S+1) \\ + \sum_{l_i \in J} a_j \cdot l^J_{t,j} \geq x_{l,t'} \geq \sum_{l_i \in J} a_j \cdot l^J_{t,j}\end{aligned} \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p\} \tag{A.13}$$

$$x_{l,t'} \geq \sum_{l_i \in J} (S+1) \cdot d^J_{t,j} \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p\} \tag{A.14}$$

$$\sum_{w_i \in I} \sum_{s=1}^{S} l^I_{t',i,s} + \sum_{l_j \in J} l^J_{t',j} \leq 1 - \sum_{w_i \in I} d^I_{t,i} \qquad \forall t \in \{0, \ldots, T-p\}, t' \in \{t, \ldots, t+p-1\}, \tag{A.15}$$

$$0 \leq \sum_{t'=0}^{t} \left(l^I_{t'} - \sum_{w_i \in I} \sum_{s=1}^{S} d^I_{t',i,s} \right) \leq 1 \qquad \forall t \in \{0, \ldots, T\} \tag{A.16}$$

$$\begin{aligned}0 \leq \sum_{t'=0}^{t} \Bigg(\sum_{w_i \in I} \sum_{s=1}^{S} l^I_{t',i,s} + \sum_{l_j \in J} l^J_{t',j} \\ - \sum_{w_i \in I} d^I_{t',i} - \sum_{l_j \in J} d^J_{t',j} \Bigg) \leq 1\end{aligned} \qquad \forall t \in \{0, \ldots, T\} \tag{A.17}$$

$$0 \leq \sum_{t'=0}^{t} \sum_{w_i \in I} \left(\sum_{s=1}^{S} l^I_{t',i,s} - d^I_{t',i} \right) \leq 1 \qquad \forall t \in \{0, \ldots, T\} \tag{A.18}$$

$$0 \leq \sum_{t'=0}^{t} \sum_{l_j \in J} \left(l^J_{t',j} - d^J_{t',j} \right) \leq 1 \qquad \forall t \in \{0, \ldots, T\} \tag{A.19}$$

$$\sum_{t=0}^{T} \left(\sum_{s=1}^{S} l^I_{t,i,s} + d^I_{t,i,s_i} \right) = 1 \qquad \forall w_i \in I \tag{A.20}$$

$$\sum_{t=0}^{T} \left(d^I_{t,i} + d^I_{t,i,s_i} \right) = 1 \qquad \forall w_i \in I \tag{A.21}$$

$$\sum_{t=0}^{T} l^I_{t,i,s} \leq \sum_{t=0}^{T} d^I_{t,i,s} \qquad \forall w_i \in I, s \in \{1,\ldots,S\} \tag{A.22}$$

$$(1 - \sum_{t=0}^{T} l^I_{t,i,s}) \cdot T + \sum_{t=0}^{T} t \cdot l^I_{t,i,s} \geq \sum_{t=0}^{T} t \cdot d^I_{t,i,s} \qquad \forall w_i \in I, s \in \{1,\ldots,S\} \tag{A.23}$$

$$\sum_{t=0}^{T} t \cdot d^I_{t,i} \geq \sum_{t=0}^{T} \sum_{s=1}^{S} t \cdot l^I_{t,i,s} \qquad \forall w_i \in I, \tag{A.24}$$

$$\sum_{t=0}^{T} t \cdot d^J_{t,j} \geq \sum_{t=0}^{T} t \cdot l^J_{t,j} \qquad \forall l_j \in J, \tag{A.25}$$

$$(T+1) \cdot u_j \geq C - d_j + 0.5 \qquad \forall l_j \in J, \tag{A.26}$$

$$u_j \leq \sum_{i=j+1}^{j+\lambda_{j+1}} v_i \qquad \forall l_j \in J, \tag{A.27}$$

$$q_j \geq 0.5 \cdot (u_j + v_j) \qquad \forall l_j \in J, \tag{A.28}$$

$$\sum_{t=0}^{T} d^J_{t,j} \geq q_j \qquad \forall l_j \in J, \tag{A.29}$$

$$q_j \cdot r_j \leq \sum_{t=0}^{T} t \cdot d^J_{t,j} + p \leq d_j \qquad \forall l_j \in J, \tag{A.30}$$

$$l^I_t \in \{0,1\} \qquad \forall t \in \{0,\ldots,T\}, \tag{A.31}$$

$$l^I_{t,i,s}, d^I_{t,i,s} \in \{0,1\} \qquad \forall t \in \{0,\ldots,T\}, s \in \{1,\ldots,S\}, w_i \in I, \tag{A.32}$$

$$l^J_{t,j}, d^J_{t,j} \in \{0,1\} \qquad \forall t \in \{0,\ldots,T\}, l_j \in J, \tag{A.33}$$

$$d^I_{t,i} \in \{0,1\} \qquad \forall t \in \{0,\ldots,T\}, w_i \in I, \tag{A.34}$$

$$x_{c,t} \in \mathbb{R}^+_0 \qquad \forall c \in \{w,l\}, t \in \{0,\ldots,T\}, \tag{A.35}$$

$$C \in \mathbb{R}^+_0, \tag{A.36}$$

$$u_j, v_j, q_j \in \{0,1\} \qquad \forall l_j \in J, \tag{A.37}$$

$$v_j = 0 \qquad \forall j \in \{m+1,\ldots,m+\lambda_m\}. \tag{A.38}$$

Note that, in contrast to Briskorn et al. (2016), we explicitly allow for the landside crane to move left while being loaded with a seaside container. Objective (A.1) minimizes the seaside makespan C, where C is set to a value not smaller than the time instant when the last seaside job has been dropped in its target slot by one of the cranes in conditions (A.2) and (A.3). Restrictions (A.4) fix the initial locations of the cranes. Constraints (A.5) and (A.6) guarantee that each crane can move at most one slot in one time unit and assure that the cranes do not cross, respectively. Conditions (A.7) restrict the cranes to move within the block, i.e set the rightmost slot to $S+1$. Conditions (A.8) constrain the seaside crane to drop the seaside containers in the given sequence. Restrictions (A.9)–(A.14) guarantee that the cranes are actually located in the slots where they lift and drop containers during the entire process of lifting or dropping. Inequalities (A.15) ensure that the landside crane does not simultaneously lift and drop a container in the same slot. Conditions (A.16)–(A.19) ensure that the difference of the number of containers that have been lifted and dropped by a crane can be at most one at any time instant.

The fact that each seaside container must reach its target slot and that each seaside container may be handled at most once by each crane is represented by restrictions (A.20) and (A.21). Conditions (A.22) and (A.23) ensure that handover containers can only be picked up in a specific slot after they have been dropped in this very slot by the seaside crane. Constraints (A.24) and (A.25) enforce the landside crane to lift containers before they can be dropped. The system of restrictions (A.26)–(A.30) guarantees that those landside containers that must be delivered at the landside handover point are dropped in their respective time windows. Finally, (A.31)–(A.38) define the domains of the variables.

Appendix B. Proofs of the Dominance Properties

Appendix B.1. Proof of Property 1

Assume that we are given an optimal schedule OPT with the seaside crane being unloaded and moving right, i.e. $x_{w,t-1} = x_{w,t} - 1$, in some period t (the waiting case $x_{w,t-1} = x_{w,t}$ is analogous). If $y_{i,t-1} > 0$ for all $i \in \{1, \ldots, n\}$, the seaside crane has already processed all seaside containers in the given schedule OPT and, because it may only process each container once, we can modify OPT by making the seaside crane move left in periods t to $t + x_{w,t-1} - 1$ (or wait in slot $s = 0$ if $x_{w,t-1} = 0$) and afterwards wait in slot $s = 0$ without interfering with the landside crane's operations and without influencing the makespan of OPT. Similarly, if there exists a seaside container $w_{\hat{i}}$ with $\hat{i} = \min\{i | y_{i,t-1} = 0\}$ in OPT, then $w_{\hat{i}}$ will necessarily be the next container to be lifted by the seaside crane in OPT. Let us refer to the time instant in which the crane has finished lifting $w_{\hat{i}}$ in OPT as $\hat{t}$. Then we can construct an optimal schedule OPT$'$ by modifying OPT as follows. The seaside crane moves left in periods t to $t + x_{w,t-1} - 1$ (or, if $x_{w,t-1} = 0$, starts lifting container $w_{\hat{i}}$), and afterwards lifts $w_{\hat{i}}$ and waits in slot $s = 0$ until time instant $\hat{t}$. The remainder of OPT$'$ corresponds to OPT. Obviously, this does not interfere with the landside crane's operations and does not change the makespan.

Appendix B.2. Proof of Property 2

Assume that we are given an optimal schedule OPT with the seaside crane being loaded with container $w_i \in I$ and moving left, i.e. $x_{w,t-1} = x_{w,t} + 1$, in some period t. Denote the very time instant in which the seaside crane has finished picking up container w_i in OPT by $\bar{t} < t$ and let $\hat{t} > t$ be the time instant in which w_i is dropped off in its target slot s_i by the seaside crane in OPT. Then we can construct an optimal schedule OPT$'$ by modifying the seaside crane's operations in OPT as follows. After having picked up container w_i in $\bar{t}$, the seaside crane waits in slot 0 until time instant $\hat{t} - p - s_i$ and afterwards keeps moving right until it has reached the target slot s_i, where it immediately drops container w_i. The landside crane's movements in OPT remain feasible in OPT$'$ because the seaside crane reaches every slot $0 < s \leq s_i$ at the latest possible time instant that allows to finish dropping w_i at $\hat{t}$, which is feasible by assumption. Additionally, the makespan does not increase because the process

of dropping w_i ends exactly at the same time instant $\hat{t}$ as in OPT. The same reasoning holds for w_i being a handover container in OPT.

Appendix B.3. Proof of Property 3

Assume that we are given an optimal schedule OPT with the seaside crane being loaded with container $w_i \in I$ and being located at the container's target slot at time instant $t-1$ ($x_{w,t-1} = s_i$). First, consider the case of w_i being dropped in its target slot s_i by the seaside crane at time instant $\hat{t} > t-1+p$ of OPT. Due to Property 2 we may assume that, in this case, the seaside crane waits in slot s_i from time instant $t-1$ to time instant $\hat{t}-p$ before it starts dropping the container in OPT. We can construct an optimal schedule OPT$'$ by modifying OPT with the seaside crane dropping w_i in s_i in time intervals t to $t-1+p$ and then proceeding in line with Property 1 until $\hat{t}$. Now assume that w_i is a handover container in OPT. Due to Property 2 we may assume that the corresponding handover slot is to the right of s_i. Let $\bar{t}$ be the time instant at which w_i is dropped in this handover slot. Then we can construct an optimal schedule OPT$''$ by modifying the seaside crane's operations in OPT as follows. After having picked up container w_i, the seaside crane waits in slot 0 until time instant $\bar{t}-p-s_i$ and afterwards keeps moving right to slot s_i, where it immediately drops w_i. The landside crane's movements in OPT remain feasible in OPT$''$ and the makespan does not increase, because the process of dropping w_i is finished at time instant $\bar{t}$ as in OPT with the seaside crane being located further to the left and w_i having arrived in its target slot.

Appendix B.4. Proof of Property 4

If $x_{l,t-1} \neq x_{l,t}+1$ in OPT, we can modify OPT by making the landside crane move left in period t and adjust its subsequent operations accordingly. Consider time slot t of the given schedule OPT. First, note that all seaside crane movements in OPT remain feasible in the modified schedule due to assumption 1. Now assume that the next container to be processed by the landside crane in OPT is any seaside container. In the modified schedule, the landside crane will be able to process the container in the very same time intervals because of assumption 2. Next, assume that the next container to be processed by the landside crane in OPT is a landside container $l_j \in J$. The earliest possible time instant to drop this container in slot $S+1$ in OPT is r_j. Hence, the unloaded landside crane needs not lift l_j in time interval t, if $t \leq r_j - (S+1-z_{j,t-1}+2p)$. Thus, the leftmost slot where the crane may be located at time instant t to be guaranteed to lift l_j on time is slot $\max\{1, z_{j,t-1} - [r_j - (S+1-z_{j,t-1}+2p) - t]\}$, which corresponds to assumption 3.

Appendix B.5. Proof of Property 5

Assume that we are given an optimal schedule OPT with the landside crane being loaded with a landside container $l_j \in J$ at time instant $t-1$ and moving left in time slot t, i.e. $x_{l,t-1} = x_{l,t}+1$ (the case $x_{l,t-1} = x_{l,t}$ is analogous). Obviously, the earliest possible time instant $\hat{t} > t$ to arrive in

the container's target slot in OPT is $\hat{t} = t + S + 2 - x_{l,t-1}$. Now, if $x_{l,t-1} \neq S+1$, we can construct an optimal schedule OPT$'$ by modifying OPT such that the landside crane moves right in time slot t, which induces the crane to be able to arrive in slot $S+1$ at $t + S - x_{l,t-1} < \hat{t}$. Hence, the makespan does not increase while all seaside crane movements remain feasible. The same holds for the case that $x_{l,t-1} = S+1$, where we can modify OPT such that the landside crane waits in $S+1$ or, if additionally $r_j \leq t - 1 + p$, immediately drops the container.

Appendix B.6. Proof of Property 6

Assume that we are given an optimal schedule OPT with the landside crane being loaded with a seaside container $w_i \in I$ and waiting in time slot t, i.e. $x_{l,t-1} = x_{l,t}$, while $x_{w,t} < x_{l,t} - 1$. Assume that the landside cranes drops w_i in slot $x_{l,t-1}$ at $\hat{t}$ of OPT without ever moving away from $x_{l,t-1}$ in between $t-1$ and $\hat{t}$. In this case, we can alternatively make the crane drop the container in time slots t to $t-1+p$ and afterwards wait in slot $x_{l,t-1}$ (or proceed in line with Property 4) until $\hat{t}$ while being unloaded without changing the makespan and without interfering with the seaside crane's movements in OPT. Now assume that the landside crane moves right from slot $x_{l,t-1}$ to $x_{l,t-1}+1$ in time slot $[\bar{t}, \bar{t}+1]$ with $\bar{t} \geq t$ of OPT while still being loaded with w_i. Then we can alternatively make the crane move right in time slot t and adjust its subsequent operations accordingly without influencing the makespan and without interfering with the seaside crane's movements in OPT. Hence, we are left with the case of assuming that the landside crane is waiting in slot $x_{l,t-1}$ in time slot t of OPT in order to move left to slot $x_{l,t-1} - 1$ while being loaded with w_i in some time slot $[t', t'+1]$ with $t' \geq t$. Then, because of Property 3, we can construct an optimal schedule OPT$'$ by modifying OPT as follows: the landside crane delays lifting w_i while waiting unloaded in the corresponding origin slot, such that the seaside crane is located in the neighboring slot when the process of lifting w_i is completed. Both cranes then mutually move right, until the seaside crane reaches the relevant destination slot. The landside crane waits in the neighboring slot while the seaside crane is dropping its container to later move left loaded.

Appendix B.7. Proof of Property 7

Consider an optimal solution OPT to an instance of PSCP-SL in which the landside crane moves left while being loaded with a container $w_i \in I$. Because of Property 3, we can assume w.l.o.g. that w_i is picked up by the landside crane in a slot left of s_i in OPT. Hence, we can furthermore assume that the landside crane continuously moves right and does not wait after having picked up the container, until it reaches s_i. Now assume the only non-trivial case, i.e. the case where the landside crane must additionally give way to the seaside crane in OPT by moving even further to the right in order to achieve the optimal seaside makespan, while the seaside crane is loaded with a container $w_{i'} \in I$ that it drops in slot s' in OPT. Then it can be seen that it is optimal for both cranes to move right together until the landside crane has reached slot $s'+1$, where it waits for the seaside crane to drop the container.

Afterwards, both cranes keep moving left (Property 1), until the landside crane has reached s_i, where it can now drop w_i without interfering with any of the seaside crane' succeeding movements in OPT.

Appendix B.8. Proof of Properties 8 and 9

An optimal schedule in which a lifting or dropping operation is interrupted for a given number t' of time periods can be modified in line with the other properties such that, for example, the start of the lifting or dropping operation is postponed by t' time periods and is then executed without interruption. Similarly, if a lifting operation is preceded by waiting for a given number of $\bar{t}$ time periods, one can modify the given schedule in line with the other properties, e.g. by making the crane enter the corresponding storage slot $\bar{t}$ time periods later.

Appendix C. Detection of Interferences

Consider a state $\mathfrak{s} = (t, x_{w,t}, x_{l,t}, active, container, c^w, list^h, list^l)$ and a given assignment of jobs to the cranes, where job^l and job^w denote l's and w's job, respectively. Furthermore, define

- s_{start_l}: slot, where l must begin lifting or continue processing (if active) the container that corresponds to job^l
- s_{stop_l}: destination slot of the container that corresponds to job^l
- s_{stop_w}: destination slot defined by job^w

Our procedure to detect an interference is summarized in Algorithm 3. It is based on the properties of Section 3, expects a state $\mathfrak{s}$ and the job assignment job^l and job^w as input parameters, and returns *true* in case of an interference. Basically, the algorithm assumes that there are no non-crossing constraints and checks if the resulting crane movements cause a collision.

Algorithm 3 (Detect Interference)

procedure DETECTINTERFERENCE($\mathfrak{s}, job^l, job^w$) ▷ *Note: s_{stop_w} is known by definition of job^w*
  if $job^w = \emptyset$ OR $job^l = \emptyset$ **then**
    return *false* ▷ *If one of the cranes has no job: no interference*
  end if
  ▷ *1. determine $t^l_{start_l}$, $t^l_{stop_l}$, s_{start_l}, and s_{stop_l}*
  if $active = l$ **then**
    if $container = (\ldots, \uparrow, p')$ **then**
      $t_{operation} := p'$
    else
      $t_{operation} := 0$
    end if
    $t^l_{start_l} := t + t_{operation}$ ▷ *time instant before l can leave s_{start_l}*
    $s_{start_l} := x_{l,t}$
    if $container = (\ldots, \downarrow, p')$ **then**
      $t_{operation} := p'$
    else
      $t_{operation} := p$
    end if

if $container = (w_i, \dots, \dots)$ **then**
$t^l_{stop_l} := t^l_{start_l} + t_{operation} + |s_i - x_{l,t}|$ ▷ *time instant before l can leave s_{stop_l}*
$s_{stop_l} := s_i$
else
$t^l_{stop_l} := t^l_{start_l} + t_{operation} + (S + 1 - x_{l,t})$
$s_{stop_l} := S + 1$
end if
else ▷ *$active = w$ or $active = \emptyset$*
if $job^l = w_i$ **then**
$t^l_{start_l} := t + |x_{l,t} - h_i| + p$
$s_{start_l} := h_i$
$t^l_{stop_l} := t^l_{start_l} + (s_i - h_i) + p$
$s_{stop_l} := s_i$
else ▷ *$job^l = l_j$*
$t^l_{start_l} := t + |x_{l,t} - a_j| + p$
$s_{start_l} := a_j$
$t^l_{stop_l} := t^l_{start_l} + (S + 1 - a_j) + p$ ▷ *r_j doesn't need to be considered for detecting interferences*
$s_{stop_l} := S + 1$
end if
end if

▷ *If the slot that l must move to in order to begin lifting or continue processing job^l is to the right of the destination slot of job^l: no interference (Properties 3, 6, 7).*

▷ *If, for all remaining cases, the destination slot of job^w is to the left of the slot that l must move to in order to begin lifting or continue processing job^l: no interference.*

if $s_{start_l} > s_{stop_l}$ OR $s_{stop_w} < s_{start_l}$ **then**
return $false$
end if
▷ *2. determine $t^{w,in}_{start_l}$, and $t^{w,in}_{stop_l}$*
if $active = w$ **then**
if $container = (\dots, \uparrow \vee \downarrow, p')$ **then**
if $s_{start_l} > x_{w,t}$ **then**
$t_{operation} := p'$
else
$t_{operation} := -p + p'$
end if
else
if $s_{start_l} > 0$ **then**
$t_{operation} := 0$
else ▷ *l is assigned a seaside container that has not yet been dropped by w*
$t_{operation} := -p$
end if
end if
$t^{w,in}_{start_l} := t + t_{operation} + (s_{start_l} - x_{w,t})$ ▷ *time instant when w has entered or can enter slot s_{start_l}*
if $s_{stop_l} \leq s_{stop_w}$ **then**
if $container = (\dots, \uparrow, p')$ **then**
$t_{operation} := p'$
else if $container = (\dots, \downarrow, p')$ **then**
$t_{operation} := -p + p'$
else
$t_{operation} := 0$
end if
else

$t_{operation} := \bar{p}(\mathfrak{s})$
end if
$t^{w,in}_{stop_l} := t + t_{operation} + (s_{stop_l} - x_{w,t})$ ▷ *time instant when w has entered or can enter slot s_{stop_l}*
else ▷ *active $= l$ or active $= \emptyset$*
if $s_{start_l} = 0$ **then**
$t^{w,in}_{start_l} := t + x_{w,t}$
else
$t^{w,in}_{start_l} := t + x_{w,t} + p + s_{start_l}$
end if
if $s_{stop_w} < s_{stop_l}$ **then**
if $s_{start_l} = 0$ **then**
$t^{w,in}_{stop_l} := t^{w,in}_{start_l} + (s_{stop_l} - s_{start_l}) + 2p$
else
$t^{w,in}_{stop_l} := t^{w,in}_{start_l} + (s_{stop_l} - s_{start_l}) + p$
end if
else
$t^{w,in}_{stop_l} := t^{w,in}_{start_l} + (s_{stop_l} - s_{start_l})$
end if
end if
▷ *If the destination slot of job^w is left of the destination slot of job^l and if l can leave s_{start_l} before w enters this slot: no interference*
▷ *If the destination slot of job^w is equal to or is located to the right of the destination slot of job^l and if l can leave s_{start_l} and s_{stop_l} before w enters these slots: no interference*
if $s_{stop_w} < s_{stop_l}$ AND $t^l_{start_l} < t^{w,in}_{start_l}$ **then**
return $false$
else if $s_{stop_w} \geq s_{stop_l}$ AND $t^l_{start_l} < t^{w,in}_{start_l}$ AND $t^l_{stop_l} < t^{w,in}_{stop_l}$ **then**
return $false$
end if
▷ *3. check whether there is an interference between s_{start_l} and s_{stop_w}*
$t^{l,in}_{stop_w} := t + (x_{l,t} - s_{stop_w})$ ▷ *time instant when l has entered or can enter slot s_{stop_w}*
if $s_{stop_w} < s_{stop_l}$ **then**
$t^{l,out}_{stop_w} := t^{l,in}_{stop_w} + 2(s_{stop_w} - \min\{s_{start_l}, x_{l,t}\}) + p$ ▷ *time instant before l can leave slot s_{stop_w}*
else
$t^{l,out}_{stop_w} := t^{l,in}_{stop_w} + 2(s_{stop_w} - \min\{s_{start_l}, x_{l,t}\}) + 2p$
end if
if $active = w$ **then**
if $container = (\ldots, \uparrow, p')$ **then**
$t_{operation} := p'$
else if $container = (\ldots, \downarrow, p')$ **then**
$t_{operation} := -p + p'$
else
$t_{operation} := 0$
end if
$t^{w,in}_{stop_w} := t + t_{operation} + (s_{stop_w} - x_{w,t})$ ▷ *time instant when w has entered or can enter slot s_{stop_w}*
$t^{w}_{stop_w} := t + (s_{stop_w} - x_{w,t}) + \bar{p}(\mathfrak{s})$ ▷ *time instant when w finishes dropping its container*
else
$t^{w,in}_{stop_w} := t + x_{w,t} + s_{stop_w} + p$
$t^{w}_{stop_w} := t^{w,in}_{stop_w} + p$
end if
▷ *Interference scenarios:*
▷ *If l is active, then there must be an interference, as otherwise the procedure would already have returned false.*

▷ *In all remaining cases, we have to check if there is an interference between* s_{start_l} *and* s_{stop_w}.
if *active* $= l$ OR ($t^{w}_{stop_w} \geq t^{l,in}_{stop_w}$ AND $t^{l,out}_{stop_w} \geq t^{w,in}_{stop_w}$) **then**
▷ *Additionally: Calculation of relevant time instants and waiting slots for dissolving interferences in DP procedure (not depicted for the sake of brevity).*
return *true*
else
▷ *In all remaining cases: no interference.*
return *false*
end if
end procedure

If an interference is detected, we generate two succeeding states with each crane being prioritized in one of the states. The only exception is the case where the landside crane is assigned a seaside container that has not yet been dropped by the seaside crane. In this case, the seaside crane has priority when generating the succeeding state.

Appendix D. Beam Search

The following implementation of the beam search policy uses a grouped queue Q as data structure. Each group in Q includes states of a distinct level and has a fixed maximum size β. Algorithms 4 and 5 illustrate our implementation of the *push* and *pop* methods for BS. Both methods share a variable *lastLevPop* (with initial value -1) that corresponds to the next level of states that will be added to the policy.

Algorithm 4 (Beam search: push)

procedure PUSH(*successors*)
levelOfStates := *lastLevPop* + 1
for all $\mathfrak{s}$ in *successors* **do**
Let $\mathfrak{G}$ be the group of states representing level *levelOfStates* in Q
$i := \mathfrak{G}.length()$
for $j := 0$ to $\mathfrak{G}.length() - 1$ **do**
$\mathfrak{s}' := \mathfrak{G}.get(j)$ ▷ $\mathfrak{s}'$ *is the* $(j+1)$*-th element of* $\mathfrak{G}$.
if $prog(\mathfrak{s})/t > prog(\mathfrak{s}')/t'$ **then**
$i := j$
break
end if
end for
$\mathfrak{G}.insert(i, \mathfrak{s})$ ▷ *Insert* $\mathfrak{s}$ *at position* i *of* $\mathfrak{G}$. *The length of* $\mathfrak{G}$ *increases by one.*
if $\mathfrak{G}.length() > \beta$ **then**
$\mathfrak{G}.remove(\mathfrak{G}.length() - 1)$ ▷ *Remove last element of* $\mathfrak{G}$ *if it includes more than* β *elements.*
end if
end for
end procedure

Algorithm 5 (Beam search: pop)

procedure POP
if $Q.empty()$ **then**
return $\emptyset$
end if

```
    𝔊 := Q.front()
    lastLevPop := level represented by group 𝔊
    𝔰 := 𝔊.pop()                          ▷ The first element 𝔰 of 𝔊 is deleted from 𝔊.
    if 𝔊.empty() then
        Q.pop()                           ▷ An empty group is deleted from Q.
    end if
    return 𝔰
end procedure
```