Optimizing the Slab Yard Planning and Crane Scheduling Problem using a Two-Stage Approach

Anders Dohn(adh@imm.dtu.dk), Jens Clausen

Department of Management Engineering
Technical University of Denmark

November 18, 2008

Contents

1 Abstract

In this paper, we present The Slab Yard Planning and Crane Scheduling Problem. The problem has its origin in steel production facilities with a large throughput. A slab yard is used as a buffer for slabs that are needed in the upcoming production. Slabs are transported by cranes and the problem considered here, is concerned with the generation of schedules for these.

The problem is decomposed and modeled in two parts, namely a planning problem and a scheduling problem.

In the planning problem a set of crane operations is created to take the yard from its current state to a desired goal state. The aim of the planning problem is twofold. A number of compulsory operations are generated, in order to comply with short term planning requirements. These operations are mostly moves of arriving and leaving slabs in the yard. A number of non-compulsory operations with a long term purpose are also created. A state of the yard may be more or less suited for future operations. It is desirable to keep the yard in a state, where it lends itself well to the future requests. Partial knowledge of future requests may exist and hence the yard can be prepared for those.

In the scheduling problem, an exact schedule for the cranes is generated, where each operation is assigned to a crane and is given a specific time of initiation. For both models, a thorough description of the modeling details is given along with a specification of objective criteria. Variants of the models are presented as well.

Preliminary tests are run on a generic setup with artificially generated data. The test results are very promising. The production delays are reduced significantly in the new solutions compared to the corresponding delays observed in a simulation of manual planning.

The work presented in this paper is focused on a generic setup. In future research, the model and the related methods should be adapted to a practical setting, to prove the value of the proposed model in real-world circumstances.

2 Introduction

2.1 Problem Description

The Slab Yard Planning and Crane Scheduling Problem is a complex optimization problem, combining planning and scheduling in an effort to generate feasible schedules for a number of interacting cranes. The problem instances origin from real world data. Costs and constraints have been defined in cooperation with the industry. The industrial problem instances are of a large size and hence it is important to create a solution method that can make superior heuristic choices in little time.

The problem at hand is from a steel hot rolling mill. A large number of slabs arrive by train to a slab yard, where they are stored until transported to the hot rolling mill by a roller table. The slabs need to be transported from the train to the yard and later from the yard to the roller table in the correct order and at specific points in time. Each slab has its individual properties and hence we need to consider each slab as being unique.

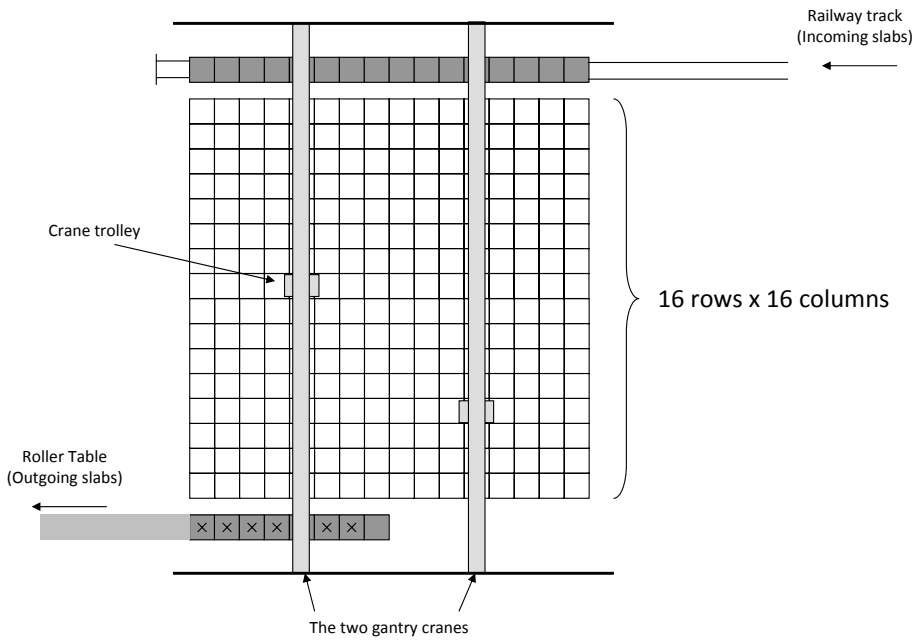

Figure 1: Overview of the slab yard.

In Figure 1 an overview of the slab yard is shown. The two gantry cranes are used to move slabs from one stack to another. As seen in the figure, both the train and the roller table can be modeled as special sets of temporary stacks. The generic slab yard that we consider in this paper consists of 16×16 stacks where each stack is a number of slabs on top of each other. The cranes can carry only one plate at a time and hence only the top slab of a stack can be moved. The cranes operate in two directions. Horizontally, they run on a shared pair of tracks and hence they can never pass each other in this direction. Vertically,

they operate by a trolley attached to the crane, which can move freely from top to bottom.

2.2 Solution Approach

The problem is approached in a two-stage planning/scheduling conception. The planning problem of the yard is of an abstract nature. Decisions are made on the layout of the slab yard. For a pre-specified time horizon, a desired end state is formulated, i.e. the end position of the slabs in the yard is determined. We also include information on a number of operations that need to be carried out in order to arrive at that state. The aim of the crane scheduling problem is to concretize the decisions of the planning problem. Operations are allocated to cranes and all operations are sequenced and positioned in time. The concrete scheduling solution is directly applicable in practice.

2.3 Literature

The Slab Yard Planning and Crane Scheduling Problem was considered in a similar context by Hansen [9]. The problem is from a Steel Shipyard where ships are constructed by welding together steel plates. The plates are stored in stacks in a plate storage and are moved by two gantry cranes sharing tracks. A simulator and a control system are developed and implemented in a system to be used as decision support for the crane operators. Our work is based on the findings of Hansen. Hansen uses a simulation as the core of the algorithm. The simulation is by nature stochastic which makes the overlying algorithm unstable. The large number of simulations needed also affects the effectiveness of the algorithm in a negative direction. For these reasons, the approach in our work is slightly different from that of Hansen.

Another similar problem is presented by Gambardella et al. [7] (based on the work in [29]) where containers are transported by cranes in a container terminal. The solution method here uses an abstract decision level and a concrete decision level to make sure that the best decisions are made and at the same time, the schedules are applicable.

An immediate advantage of the two-stage approach is that the planning problem and the scheduling problem individually have received considerable attention in the literature.

2.3.1 Slab Yard Planning and Container Stacking

The literature relating to The Slab Yard Planning Problem is found mainly in other areas of slab yard planning and in container stacking. An abstract stacking problem from the artificial intelligence literature is the Tower of Hanoi Problem, where disks are moved between stacks one by one and can only be placed on top of larger disks. Here we find some general tools for stacking problems. Dinitz and Solomon [5] describe algorithms for the Tower of Hanoi Problem with relaxed placement rules. See also the literature on Blocks World (e.g. [23]) for another abstract problem with interesting similarities to the slab stacking problem.

Tang et al. [27] describe a steel rolling mill where slabs need to be transported from a slab yard according to a scheduled rolling sequence. The article

builds on the initial work found in [26]. The layout of the slab yard is different from ours and the cranes are located so that they never collide. Another difference is that for each batch, several candidates exist among the slabs, and the objective therefore is to minimize the cost by choosing the right slabs among the candidates. Singh et al. [21] address the same problem and solve it using an improved Parallel Genetic Algorithm. König et al. [12] investigate a similar problem from storage planning of steel slabs in integrated steel production. The problem formulation in the article is kept at a general level to make the model versatile. The stacking problem is considered alone, thereby disregarding the crane schedules. They present a greedy construction heuristic and by a Linear Programming relaxation of a Mixed Integer Problem formulation of the problem, they are able to quantify the quality of their solutions.

A problem in container stacking with many similarities to the slab stacking problem is described by Dekker et al. [4]. A significant difference is that the maximum height of container stacks is 3, where the corresponding number in slab stacks usually is considerably larger than this. A number of stacking policies are investigated by means of simulation and in that sense, it resembles the work of Hansen [9] in a container stacking context. Kim and Bae [10] describe a container stacking problem where a current yard layout is given and a new desirable layout is provided. The problem is to convert the current bay layout to the desirable layout by moving the fewest possible number of containers. The problem is decomposed into three subproblems, namely a bay matching, a move planning, and a task sequencing stage, where the latter two are similar to the two stages that we introduce for The Slab Yard Planning and Crane Scheduling Problem. Kim et al. [11] consider a similar container stacking problem. See Steenken et al. [24] for at recent review of literature on container stacking.

2.3.2 Crane Scheduling

The Crane Scheduling problem considered here is an example of a Stacker Crane Problem [6] with time windows and multiple cranes.

Parallel crane/hoist scheduling has been thoroughly treated in production of electronics, especially in printed circuit board production. In circuit board production, the hoists are used to move products between tanks, where the plates are given various chemical treatments. In some layouts, the production sequence is the same for all products. One consequence of this is that all hoist moves are from one tank to the next. Such a layout is less interesting in our context.

Leung and Zhang [14] introduce the first mixed-integer programming formulation for finding optimal cyclic schedules for printed circuit board lines with multiple hoists on a shared track, where the processing sequence may be different than the location sequence of the tanks. The solution method itself is not transferable, but several of the elements in the modeling phase are very relevant to the crane scheduling problem of the slab yard. This includes the formulation of collision avoidance constraints. Collision avoidance constraints are also described in a dynamic hoist scheduling problem by Lamothe et al. [13] and in a fixed sequence production by Che and Chu [3] and Leung et al. [15].

Rodošek and Wallace [19] present an integration of Constraint Logic Programming and Mixed Integer Programming to solve hoist scheduling problems. The proposed hybrid solver is able to solve several classes of previously unsolved

hoist scheduling problems to optimality. Varnier et al. [28] also use constraint programming to find optimal hoist schedules.

A simulation module is developed by Sun et al. [25] and used to test scheduling strategies. The schedules are created in a greedy fashion, following the chosen strategy and by simulation the strategy is evaluated. Skov [22] presents a recent application in another large scale setting. Two methods are considered: a simulation approach and a method based on the alternative graph formulation (see e.g. [18] or [17]).

As it becomes apparent in the following sections we are in the scheduling problem able to abstract from the practical context of the problem and consider the problem as a parallel scheduling problem with sequence-dependent setup times. Zhu and Wilhelm [30] present a recent literature review for this type of scheduling problems.

3 Modeling

3.1 Specific Problem Properties and Assumptions

To build an accurate model we need to give a more detailed problem description.

For the storage, we assume that slabs are always placed in stacks. All stacks have a maximum height. Train wagons and the roller table are also modeled as stacks. We assume that the train has a length and loading capabilities similar to that of one row in the yard, and hence the train is modeled as a temporary row as illustrated on Figure 1. The train is only at the yard for a certain amount of time and hence all slabs must be moved away from the wagons within this time window. The modeling of the roller table is a little more complex. In principle, we have access to the roller table in multiple positions as shown on Figure 1 (shown as 8 stacks wide). The order of the slabs on the roller table is essential and to ensure that the sequence of slabs leaving the roller table follow the order in which they were brought there, we allow the cranes to bring slabs to only the right-most of the roller table stacks. Further, as there is room for at most 8 slabs on the roller table, we may have to wait, whenever the roller table is full. As time goes, the slabs are removed from the roller table in a first-in, first-out manner. For each slab that is to be moved from the yard in a near future, we have the production time, *Aim Leave Time* (*ALT*). By looking forward 8 slabs in the sequence, we know when there will be free room for a new slab on the roller table, *Earliest Leave Time* (*ELT*). Hence, we have a time window for moving the specific slab to the roller table. Slabs that have not yet been given an *ALT* (slabs which are not a part of the immediately following production) instead have an *Estimated Leave Time* (*EST*), a *Batch Identification Number* (*BID*), and a *Batch Sequence Number* (*BSQ*). As slabs are processed in batches, we know that all slabs with identical BIDs will leave the yard consecutively and in the order dictated by the BSQs of those slabs. This information can be used to prepare the yard layout for future production.

Each move consists of lift time, transportation time, and drop time. We assume that the cranes move at constant velocity. Transportation time is equal to the maximum of the vertical and the horizontal transportation time, as the cranes are able to move horizontally concurrently with the crane trolley moving in a vertical direction.

3.2 Objective

The objective of the schedule is to minimize maximum tardiness (delay). The reason is as follows. Take all slabs leaving the slab yard within the scheduling horizon. Whenever a slab is not moved to the roller table before its Aim Leave Time, it causes a delay in the production. The production is not immediately able to catch up on this delay and therefore subsequent slabs are needed later in the production than we anticipated initially. Production is further delayed, only if subsequent slabs are delayed even more. Hence, the most delayed slab determines the quality of the solution.

3.3 Decomposition

Already at an early point in the process it became clear that splitting the problem in a planning stage and a scheduling stage is valuable. In the first part, the planning stage, we make the plan of what we want to do within a given time horizon, i.e. we determine which slabs to move and where to move them. These decisions are not disturbed by specific details concerning the crane scheduling. The idea is that better decisions are made when the problem is abstracted enough to only consider the most important properties at first. Hence, it also becomes easier to express what we expect from a good solution. The scheduling stage takes care of all the details. It is important to respect all deadlines and hence doing so is the main objective in the scheduling stage. In the scheduling stage, all operations are allocated to cranes and the final order of execution is determined. A feasible schedule consists of a sequence of *Operations* in the form: Crane X picks up slab Y (at its current location) at time T and moves it to position Z. Naturally, none of the operations are allowed to conflict with other operations, neither within the schedule of one crane nor between the two cranes.

We have seen a similar decomposition of problems in the literature (e.g. [7] and [10]). Most of the related literature deals with either the yard planning problem alone or the crane scheduling problem alone. Both of these points strengthen the reasoning behind the decomposition. The decomposition and the resulting model is described in Section 4 and 5.

3.4 A Simple Example

Before describing the details of the decomposition, we introduce an illustrative example, to clarify the concepts and ideas that are introduced in the following sections.

Example 1. We have a very simple slab yard with only one row. An overview of the small yard is shown in Figure 2.

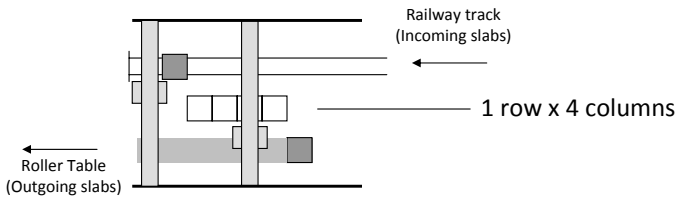

Figure 2: Overview of a very simple slab yard used in Example 1.

In this example, we have a scheduling horizon of $[0, 22]$. A side view of the initial yard state is shown in Figure 3. Note that the vertical dimension is not visible in the figure. However, in the figure, we are able to illustrate the exact composition of each stack. The yard consists of a single arrival stack, T_{ar1}, four stacks in the main yard, $T_1, ..., T_4$, and one exit stack, T_{exit}. In the yard are 14 slabs, $S_1, ..., S_{14}$.

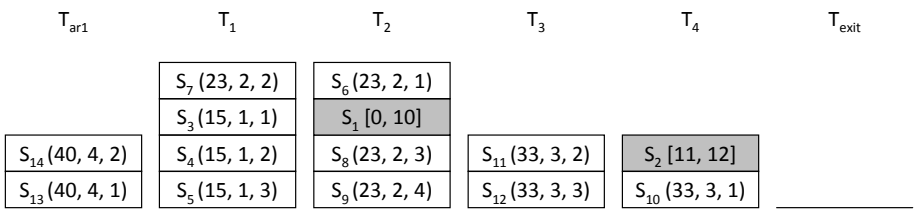

Figure 3: Slab Yard Crane Scheduling Problem: Side-view of a toy example. Gray slabs are slabs that must leave the yard during the scheduling horizon. Leaving slabs (gray): [ELT, ALT]. Non-leaving slabs (white): (EST, BID, BSQ)

4 Modeling - The Yard Planning Problem

In the planning stage we generate a plan that takes us from the current state of the yard to a final state for the horizon. In the final state, all slabs with deadline within the horizon are brought to the roller table. At the same time, the plan should leave the yard in the best possible condition for subsequent planning periods.

To arrive at a feasible and superior plan within reasonable computational time, the idea is to relax a number of the real world constraints in the planning stage. Whatever is relaxed here is fixed in the scheduling stage so that the final solution is always fully descriptive.

4.1 Operations

In the planning stage, we are going to consider a solution as defined by a number of successive operations. An operation contains the following information:

Slab The slab to be transported.

Destination The stack where the slab is put on top.

Priority How important is it to include this operation in the final schedule.

A solution to the planning problem consists of a sequence of operations. Many operations are related directly to slabs which are moved to the roller table. Such operations are compulsory and hence have a priority of ∞. As is described in Section 4.4, some operations are however optional and the priorities give an ordering of their importance.

There are three strict requirements for feasibility of The Yard Planning Problem. Furthermore, we have a number of properties that we would also like to find in a solution. Additional desired properties are described by separate functions.

4.2 Feasibility Criterion

For a planning solution to be feasible, we require the following:

- All slabs with deadline within the scheduling horizon are transported to the exit stack in the correct order.
- All incoming slabs (i.e. slabs in the train wagon stacks) must be moved to permanent stacks.
- All operations must be valid in the sequence. Only slabs on top of a stack may be moved and only to stacks where the maximum stack height has not been reached.

The two first criteria are easy to verify, when we know the set of incoming slabs and the set of outgoing slabs. The third criterion can be verified by updating a yard state as the sequence of operations is processed.

4.3 Assessment Criteria

To assess the quality of a plan, we introduce a number of objectives. The following properties characterize a good solution. The two first are directly concerned with the plan, where the two last evaluate the end state of the yard.

- The number of operations is low.
- Operations do not span too far vertically. Even though the operations are not allocated to cranes yet, we would like a solution to accommodate such an allocation. Operations are faster and have less risk of conflicting, when they span over as little vertical space as possible.
- Slabs that are to leave the yard soon (but after the current horizon) are close to the exit stack.
- The number of *false positions* is low. Slabs in false positions are slabs that are in the way of other slabs below them. A false position in our context is a stochastic term, as many slabs only have an estimated leave time. We are still able to use the notion of false positions even though it is not deterministic. As described in Section 4.3.1, we introduce probabilities to approximate the number of false positions.

All of the above points are formulated rather vaguely. To make it clear how we intend to assess the different criteria, we introduce the evaluation functions $z_1 - z_4$.

$numoperations$	=	the number of operations.
$disttoexit_s$	=	the distance from the stack of slab s to the exit stack (T_{exit}).
$leavetime_s$	=	ALT (Aim Leave Time) of slab s - if ALT does not exist use EST (Estimated Leave Time).
$falseprob_s$	=	Probability of slab s being in a false position.
$vertspan_a$	=	Vertical span of operation a.
z_1	=	$numoperations$
z_2	=	$\sum_s disttoexit_s(\max_{s'}(leavetime_{s'}) - leavetime_s)$
z_3	=	$\sum_s falseprob_s$
z_4	=	$\sum_a vertspan_a$

In a solution method we want to minimize $z_1 - z_4$. One way could be to create one multi-criterion objective function, where the four criteria are weighted and summed to give one value for the quality of the solution. $disttoexit_s$ is not necessarily the geometric distance. It may be calculated so that vertical distance weighs heavily, as movement in this direction is slower and as a large vertical distance is causing more collision avoidance inconvenience in the scheduling stage.

4.3.1 Calculating the Probability of False Positions

$falseprob_s$ is calculated in the following way. Define a set B_s of all slabs below slab s in the yard state to be evaluated. For each slab $b \in B_s$ we calculate

the probability p_b of this slab leaving the yard before s. We assume that if ALT is not defined, it follows the Gaussian distribution $alt \sim N(est, \sigma^2)$, where est is the Estimated Leave Time and σ is problem dependent. Φ_{est,σ^2} is the corresponding cumulative distribution function. We have several cases:

alt_s and alt_b are both defined:	$p_b = \begin{cases} 1 \text{ if } alt_b < alt_s \\ 0 \text{ otherwise} \end{cases}$
$bid_s = bid_b$	$p_b = \begin{cases} 1 \text{ if } bsq_b < bsq_s \\ 0 \text{ otherwise} \end{cases}$
$bid_s \neq bid_b$ and only alt_s is defined	$p_b = \Phi_{est_b,\sigma^2}(alt_s)$
$bid_s \neq bid_b$ and only alt_b is defined	$p_b = 1 - \Phi_{est_s,\sigma^2}(alt_b)$
$bid_s \neq bid_b$ and alt_s and alt_b not defined	$p_b = \Phi_{est_b - est_s, 2\sigma^2}(0)$

In the last case, we have two different Gaussian distributions for the two ALTs. We want to find the probability that $alt_b \sim N(est_b, \sigma^2)$ is smaller than $alt_s \sim N(est_s, \sigma^2)$, which is the same as finding the probability of getting a negative value from the difference between the two: $N(est_b, \sigma^2) - N(est_s, \sigma^2) = N(est_b - est_s, \sigma^2 + \sigma^2)$. The cumulative distribution function of $N(est_b - est_s, 2\sigma^2)$ is hence used in the last case.

To find the total probability we accumulate the individual probabilities. As some of the slabs in B may belong to the same batch, we may have a very high correlation between the probabilities. For simplicity, we split in two cases. If the slabs b_1 and b_2 have identical BID we say that these are totally correlated and therefore we only use one of the probabilities p_{b_1} and p_{b_2}. We choose to use $\max(p_{b_1}, p_{b_2})$. If two slabs belong to different batches we assume that the probabilities are independent and both probabilities are used in the calculation. Define the set B'_s as the set of slabs included by this selection. Now $falseprob_s$ is calculated as:

$$falseprob_s = 1 - \prod_{b \in B'_s} (1 - p_b)$$

Example 1 (continued). Solution 1

A planning solution to Example 1 is seen in Figure 4. The solution consists of a sequence of operations.

o_1: ($S_6 \rightarrow T_3$)	o_2: ($S_1 \rightarrow T_{exit}$)	o_3: ($S_2 \rightarrow T_{exit}$)	o_4: ($S_{14} \rightarrow T_2$)	o_5: ($S_{13} \rightarrow T_2$)

Figure 4: Solution 1. A solution to the planning problem of Example 1.

The end state of the storage is fully determined by the planning solution and is shown in Figure 5.

This solution may be evaluated by the objectives defined earlier. We assume

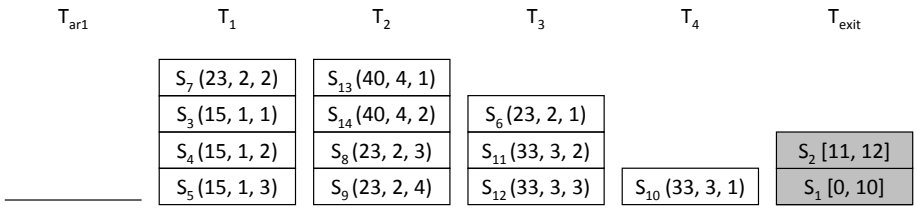

Figure 5: End state for Solution 1 (Figure 4).

$\sigma = 5$, and hence $\sigma^2 = 25$ and $2\sigma^2 = 50$. We get:

$$z_1 = numoperations = 5$$
$$z_2 = \sum_s disttoexit_s(\max_{s'}(leavetime_{s'}) - leavetime_s) = 539$$
$$z_3 = \sum_s falseprob_s = 2.9335$$
$$z_4 = \sum_a vertspan_a = 1 + 3 + 1 + 2 + 2 = 9$$

z_2 and z_3 are calculated from the following terms:

Slab	$z_2 = \sum$	$z_3 = \sum$
S_3	$4 \cdot (40 - 15) = 100$	0
S_4	$4 \cdot (40 - 15) = 100$	0
S_5	$4 \cdot (40 - 15) = 100$	0
S_6	$2 \cdot (40 - 23) = 34$	$\Phi_{10,50}(0) = 0.0786$
S_7	$4 \cdot (40 - 23) = 68$	$\Phi_{-8,50}(0) = 0.8711$
S_8	$3 \cdot (40 - 23) = 51$	0
S_9	$3 \cdot (40 - 23) = 51$	0
S_{10}	$1 \cdot (40 - 33) = 7$	0
S_{11}	$2 \cdot (40 - 33) = 14$	0
S_{12}	$2 \cdot (40 - 33) = 14$	0
S_{13}	$3 \cdot (40 - 40) = 0$	$\Phi_{-17,50}(0) = 0.9919$
S_{14}	$3 \cdot (40 - 40) = 0$	$\Phi_{-17,50}(0) = 0.9919$

Solution 2

We may alter the solution slightly by moving S_{13} and S_{14} to stack T_4 instead of T_2. This gives the solution of Figure 6.

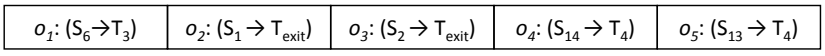

Figure 6: Solution 2. Example of a solution to the planning problem of Example 1.

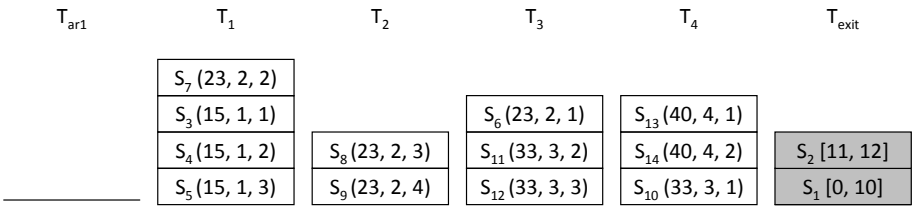

Figure 7: End state for Solution 2 (Figure 6).

The end state is only slightly changed (Figure 7). This solution gets the following evaluation:

$$z_1 = numoperations = 5$$
$$z_2 = \sum_s disttoexit_s(\max_{s'}(leavetime_{s'}) - leavetime_s) = 539$$
$$z_3 = \sum_s falseprob_s = 2.6275$$
$$z_4 = \sum_a vertspan_a = 1 + 3 + 1 + 4 + 4 = 13$$

We see that it is better in the respect of false positions. There is now a greater chance for S_{13} and S_{14} for not needing a reshuffle. On the other hand, z_4 is worse as the two new operations vertically span wider.

Solution 3
A third possible solution is shown in Figure 8.

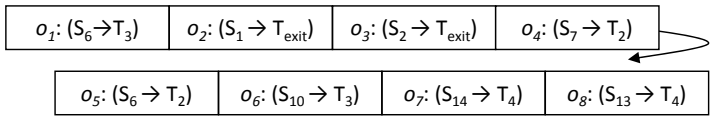

Figure 8: Solution 3. Example of a solution to the planning problem of Example 1.

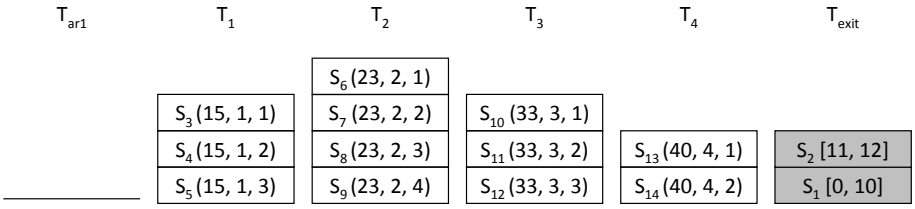

Figure 9: End state for Solution 3 (Figure 8).

This solution yields the end state of Figure 9. The end state has no possible false positions, which is naturally a good feature. On the other hand we also

need more operations in the solution. The solution is evaluated with:

$$z_1 = numoperations = 8$$
$$z_2 = \sum_s disttoexit_s(\max_{s'}(leavetime_{s'}) - leavetime_s) = 546$$
$$z_3 = \sum_s falseprob_s = 0$$
$$z_4 = \sum_a vertspan_a = 1 + 3 + 1 + 1 + 1 + 1 + 4 + 4 = 16$$

4.4 Operation Priority

So far we have assumed that all operations had to be included in the final schedule. However that need not be the case. In Solution 3 of Example 1, we saw an example, where a number of operations are added to enhance the final state. Some of the operations could be disregarded if the schedule becomes too tight. Figure 10 shows Solution 3 with priorities on the operations.

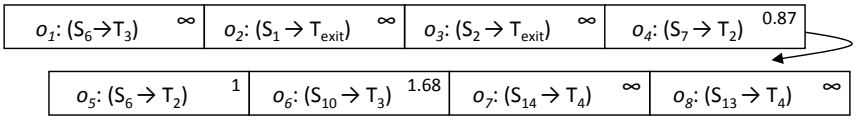

Figure 10: Solution 3. Example of a solution to the planning problem of Figure 3. The operations have priorities in the upper right corner.

The priorities of the operations in this example have been calculated as the difference in z_3 when the operation is omitted, compared to the end state where all operations are included. This priority may also be a multi-criteria function just as the objective function. As for the objective function that would require us to weigh the various criteria against each other.

Example 2. Sometimes operations may not in themselves be of any value, but they may be prerequisites of other optional operations. An example of this is seen in Figure 11 + Figure 12. The operation for S_{11} has a priority of 0, but must be included for S_{10} to be moved. Moving S_{10} adds a lot of value to the solution.

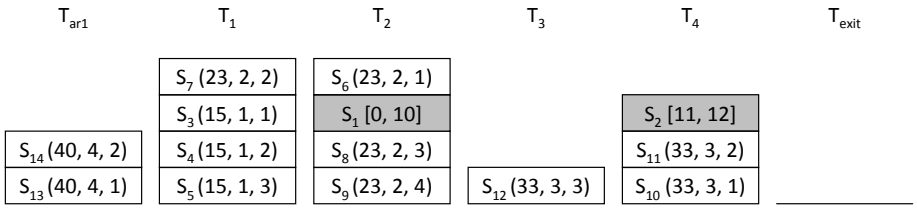

Figure 11: Example where a no-value optional operation may be included.

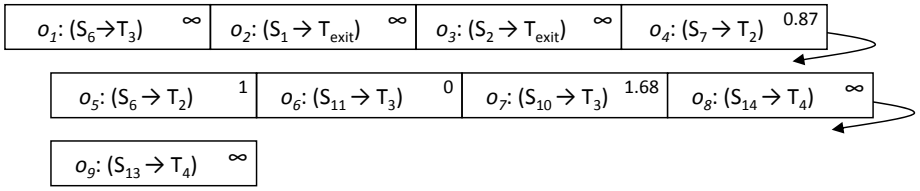

Figure 12: A solution to the problem of Figure 11.

5 Modeling - The Crane Scheduling Problem

From a solution to the planning problem it is now the aim to generate a complete and feasible schedule. First, the ordering of operations must be relaxed to allow for parallel execution of operations. Most operations are locally independent from each other. These independencies are detected and only meaningful precedence constraints are kept for the scheduler. The crane scheduling problem is similar to a traditional parallel scheduling problem. We have a number of operations that we need to allocate to two cranes (machines). Between operations there are several temporal constraints. The anti-collision constraint is an important temporal constraint added by the fact that we have two cranes in operation. As the crane operation times are of a stochastic nature, we need to introduce buffers, enforced by the temporal constraints. The buffers ensure that no crane collision occurs, even with disturbances in operation time. For major disturbances, the scheduling problem and possibly the whole planning may have to be resolved.

5.1 Precedence Relations

To ensure that the end state of the schedule is identical with end state of the planning solution, we establish a number of precedence relations. Using the planning sequence as a starting point we ensure that, whenever relevant, the order of the operations in the schedule stay the same as in the plan. There are four cases where reordering operations may change the state of the storage and may therefore cause direct or indirect infeasibility of the solution. In these cases we do not allow reordering of the operations. See Figure 13 for a graphical description of the four cases.

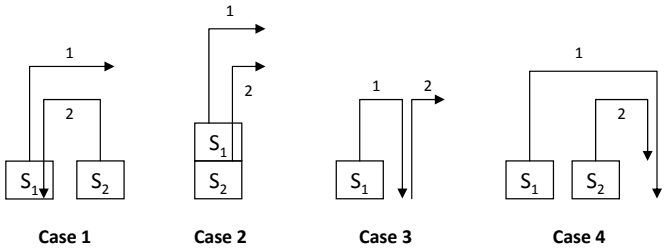

Figure 13: Graphical description of the state preserving precedences.

Case 1 Moving slab S_2 to a stack where slab S_1 was moved away from earlier. If the order of these two operations is changed, S_2 is going to block S_1 and the solution becomes infeasible. There does not seem to be much sense in changing the order of the two operations either, as it would require the addition of another move of S_2 before S_1 is moved.

Case 2 Moving slab S_1 and then slab S_2, where S_1 is on top of S_2. Again, changing the order of the two operations leads to infeasibility.

Case 3 Moving the same slab twice. If the order of such two moves is changed, the final destination of the slab may change. If the slab is moved again at a later time the final destination, however, remains unchanged.

Case 4 Two slabs S_1 and S_2 are moved to the same stack. If the order is changed it may lead to infeasibility later. If the two slabs are not moved later, the end state is altered, but the solution remains feasible.

Example 1 (continued). Going back to the Example 1, we are now able to determine the precedence relations of the plan. Using the four cases depicted in Figure 13 we arrive at the precedences in Figure 14.

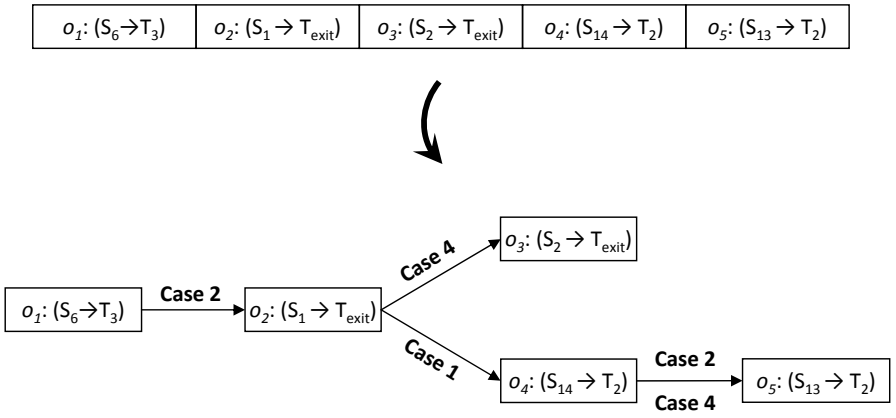

Figure 14: Precedence relations for Solution 1 (Figure 4).

5.2 Temporal Constraints

The precedence constraints described in the previous section ensure that the end state of a parallel schedule is the same as the corresponding sequential schedule. We still need to introduce temporal constraints to create a schedule that is feasible with respect to the individual movement of a crane and to create a schedule which is collision free.

5.2.1 Notation and Definitions

For two operations i and j we have four positions that have to be considered and where temporal constraints may have to be added correspondingly. The four positions are:

T_i^{orig}	Origin stack of operation i
T_i^{dest}	Destination stack of operation i
T_j^{orig}	Origin stack of operation j
T_j^{dest}	Destination stack of operation j

In the following, we say that i is before j, if i enters and leaves the conflict zone between the two moves, before j. When two operations are allocated to the

same crane, the crane needs to complete the first operation before initiating the next. Hence, in this case if i is before j it means that operation i is completed before operation j is initiated. However, if the operations are allocated to different cranes they may have a small conflict zone. Hence, even if operation i is before operation j, it does not necessarily mean that it is neither initiated first nor completed first. It only means that it will be the first of the two moves in any of their conflict positions. If two operations have no conflict zone, it is irrelevant whether i is considered to be before j or vice versa.

In the following, we calculate the required gap between two operations i and j when i is before j. The gap is defined as the amount of time required from initiation of operation i to initiation of operation j. There are three different types of gaps depending on the crane allocation of operations i and j.

g_{ij}^s	Required gap when i and j are allocated to the same crane (s).
g_{ij}^l	Required gap when i is allocated to the left crane (l) and j to the right crane.
g_{ij}^r	Required gap when i is allocated to the right crane (r) and j to the left crane.

The following generalized precedence constraint is imposed: $t_i + g_{ij} \leq t_j$, where g_{ij} represents g_{ij}^s, g_{ij}^l or g_{ij}^r according to the situation. To calculate the gaps between operations, we need to introduce a number of parameters:

p_i	time required to pick up slab of operation i.
q_i	time required to drop off slab of operation i.
$m_{T_xT_y}$	time required to move from stack T_x to stack T_y when crane is laden.
$e_{T_xT_y}$	time required to move from stack T_x to stack T_y when crane is empty.
b	buffer time required between two cranes (see Section 5.2.3).

We assume that $m_{T_xT_y}$ and $e_{T_xT_y}$ are linear in the distance traveled. Both measures are independent of the crane involved. In the following we will use the assumption that the two cranes move at the same speed. Also, we assume that a crane cannot move faster when laden than when it is empty.

Precedence relations are included in the generalized precedence constraints, so the values of g_{ij}^s, g_{ij}^l and g_{ij}^r hold all the information we need with respect to precedence constraints. If precedence relations disallow the execution of operation i before operation j, we set: $g_{ij}^s = g_{ij}^l = g_{ij}^r = \infty$.

5.2.2 Two Operations Allocated to the Same Crane

When two operations are allocated to the same crane, we need to make sure that there is sufficient time to finish the first operation and to move to the start position of the second operation. Consequently, we get:

$$g_{ij}^s = p_i + m_{T_i^{orig}T_i^{dest}} + q_i + e_{T_i^{dest}T_j^{orig}}$$

See Figure 15 for a visualization of this. We use Time-Way Diagrams that are frequently used when depicting solutions of crane scheduling problems, especially in printed circuit board production (see e.g. [16]). The horizontal and vertical axes in the diagram represent the time and crane positions, respectively.

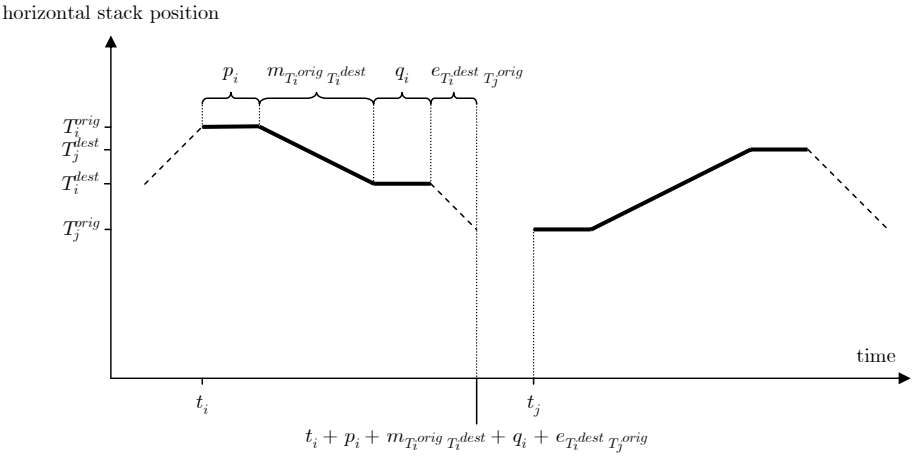

Figure 15: Two operations executed sequentially by the same crane.

5.2.3 Two Operations Allocated to Separate Cranes

When two operations are allocated to two separate cranes, we need to make sure that the cranes never collide. Further, as we are dealing with a highly stochastic system, we introduce the concept of a buffer. The buffer denotes the amount of time we require between two cranes traversing the same position. By introducing a buffer we establish a certain degree of stability in the schedule. If one of the cranes is delayed by an amount of time less than the buffer size the schedule is still guaranteed to be feasible. The buffer size is set so that infeasibility only occurs in rare cases. In the following, a violation of the prespecified buffer size is considered to be a collision.

Left Crane Moves First In Table 1 we describe how to calculate g_{ij}^l. This is the case, where the left crane is allocated to operation i and the right crane to operation j. In case of conflict between the two operations, operation i enters and leaves the conflict zone before operation j. There are five different cases to be considered. These are shown in Table 1 and in Figures 16-20. $(l2)$ and $(l3)$ may both apply at the same time and in that case g_{ij}^l is set equal to the larger of the two values. To be strict, we may rewrite $(l2)+(l3)$ as $(l2b)+(l3b)+(l23)$, see Table 2. The comparison of two stacks is done with respect to their horizontal position, e.g. $T_i^{orig} < T_j^{orig}$ means origin stack of operation i is to the left of origin stack of operation j.,

	Precondition	Gap
$(l1)$	$T_j^{orig} \leq T_i^{dest}$	$g_{ij}^l = p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b$
$(l2)$	$T_j^{dest} \leq T_i^{dest} < T_j^{orig}$	$g_{ij}^l \geq p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b - (p_j + m_{T_j^{orig} T_i^{dest}})$
$(l3)$	$T_i^{dest} < T_j^{orig} \leq T_i^{orig}$	$g_{ij}^l \geq p_i + m_{T_i^{orig} T_j^{orig}} + b$
$(l4)$	$T_i^{dest} < T_j^{dest} \leq T_i^{orig} < T_j^{orig}$	$g_{ij}^l = p_i + m_{T_i^{orig} T_j^{dest}} + b - (p_j + m_{T_j^{orig} T_j^{dest}})$
$(l5)$	Otherwise	$g_{ij}^l = -\infty$

Table 1: Calculation of the required gap between two operations. Operation i is allocated to the left crane and operation j to the right crane. In conflict, i is moved before j.
Otherwise means: $T_j^{orig} > T_i^{orig} \wedge T_j^{orig} > T_i^{dest} \wedge T_j^{dest} > T_i^{orig} \wedge T_j^{dest} > T_i^{dest}$

	Precondition $\Rightarrow$ Gap
$(l2b)$	$T_j^{dest} \leq T_i^{dest} < T_j^{orig} \wedge T_i^{orig} < T_j^{orig}$ $\Rightarrow g_{ij}^l = p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b - (p_j + m_{T_j^{orig} T_i^{dest}})$
$(l3b)$	$T_i^{dest} < T_j^{orig} \leq T_i^{orig} \wedge T_i^{dest} < T_j^{dest}$ $\Rightarrow g_{ij}^l = p_i + m_{T_i^{orig} T_j^{orig}} + b$
$(l23)$	$T_j^{dest} \leq T_i^{dest} < T_j^{orig} \leq T_i^{orig}$ $\Rightarrow g_{ij}^l = \max(p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b - (p_j + m_{T_j^{orig} T_i^{dest}}), p_i + m_{T_i^{orig} T_j^{orig}} + b)$

Table 2: We may rewrite $(l2) + (l3)$ of Table 1 as $(l2b) + (l3b) + (l23)$.

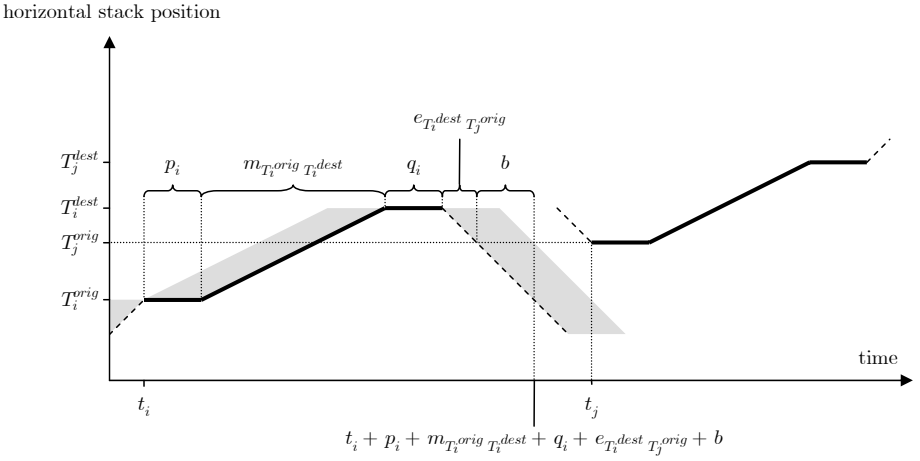

Figure 16: $(l1)$: $T_j^{orig} \leq T_i^{dest} \Rightarrow t_i + p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b \leq t_j$

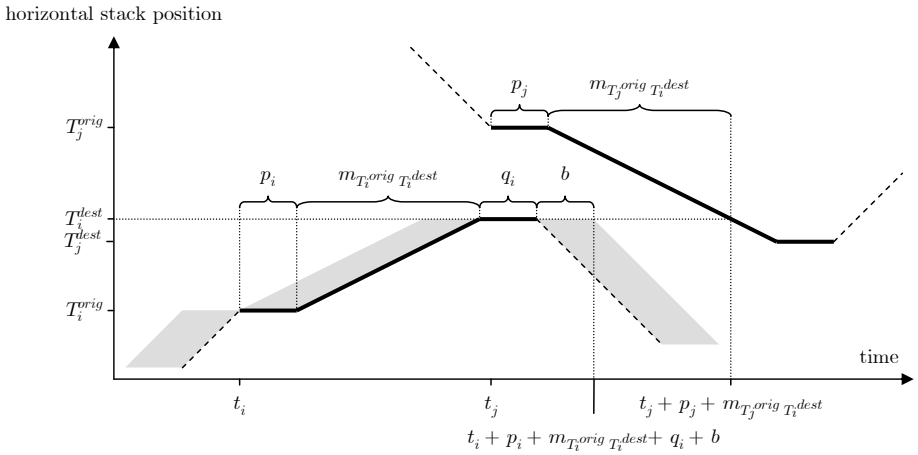

Figure 17: ($l2$): $T_j^{dest} \leq T_i^{dest} < T_j^{orig} \Rightarrow t_i + p_i + m_{T_i^{orig}T_i^{dest}} + q_i + b \leq t_j + p_j + m_{T_j^{orig}T_i^{dest}}$

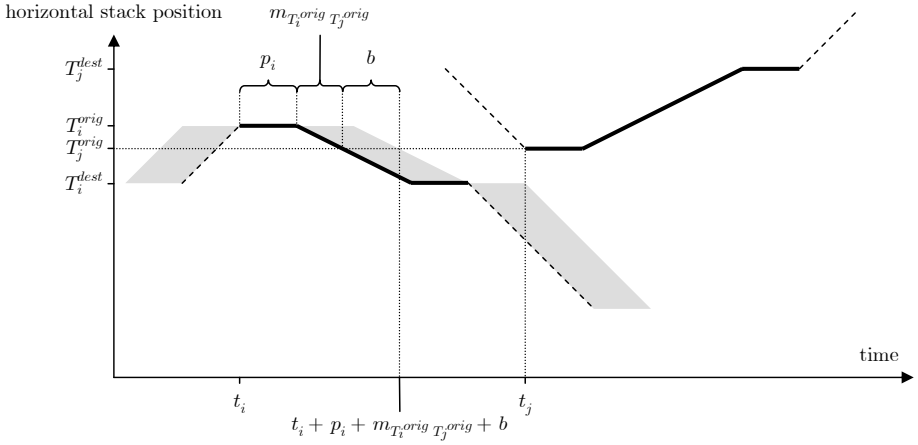

Figure 18: ($l3$): $T_i^{dest} < T_j^{orig} \leq T_i^{orig} \Rightarrow t_i + p_i + m_{T_i^{orig}T_j^{orig}} + b \leq t_j$

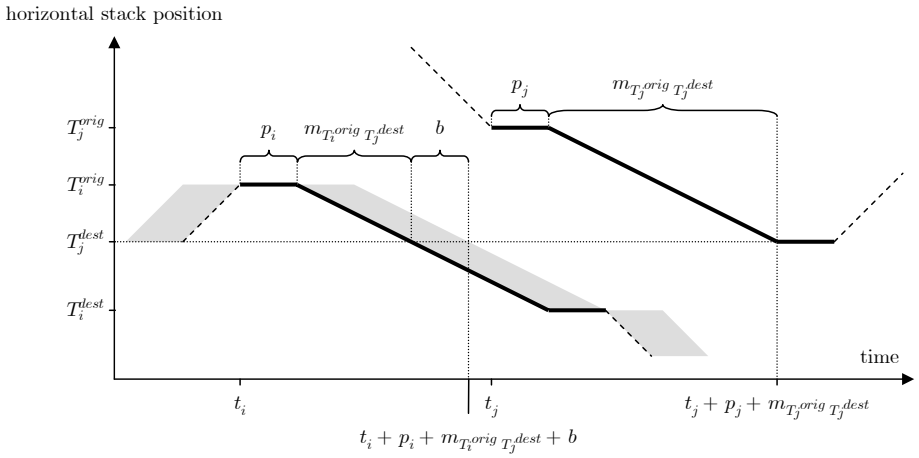

Figure 19: ($l4$): $T_i^{dest} < T_j^{dest} \leq T_i^{orig} < T_j^{orig} \Rightarrow t_i + p_i + m_{T_i^{orig}T_j^{orig}} + b \leq t_j + p_j + m_{T_j^{orig}T_j^{dest}}$

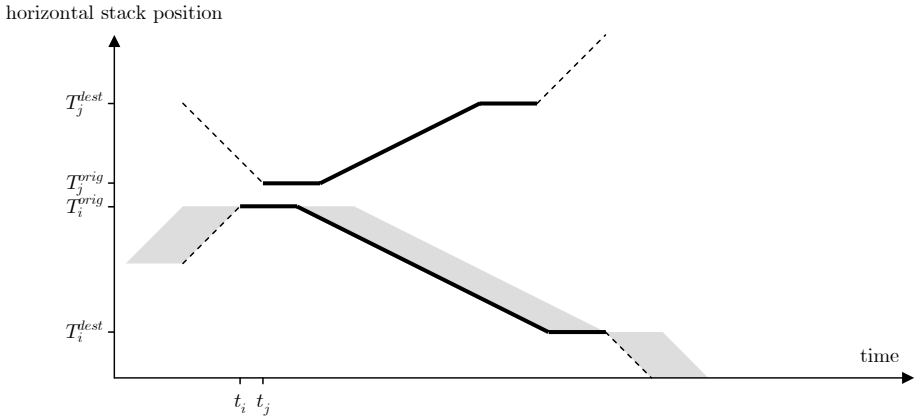

Figure 20: ($l5$): No direct temporal relations between operation i and operation j.

It should be clear from each of the five figures (Figure 16 - Figure 20) why a violation of the constraint introduces a collision. However, what may not be so clear is why these five cases are sufficient for avoiding all possible collisions. We will prove this in the following. It is important to note that we are only comparing two operations at a time. We may hence assume that the left crane is moved away as soon as it finishes and in the same way we may assume that the right crane is only moving into the overlapping area of the storage when absolutely necessary. If all pairwise comparisons show no collisions the whole schedule is collision-free.

Proposition 1. *Assuming that a crane cannot move faster when laden than when it is empty and that the two cranes move at the same speed. If the left crane is allocated to operation i and the right crane to operation j, then $(l1)-(l5)$ are sufficient to avoid all collisions between the two cranes.*

Proof. First, assume that $T_j^{orig} \leq T_i^{dest}$. The situation is depicted in Figure 21. The horizontally hatched region can never be entered by either crane as we assume the unladen movement speed to be at least as fast as the laden movement speed. This is true for all values of T_i^{orig} and T_j^{dest}. The region is at least as wide as the buffer because $t_j - (t_i + p_i + m_{T_i^{orig}T_i^{dest}} + q_i + e_{T_i^{dest}T_j^{orig}}) \geq b$ is ensured by $(l1)$. Hence, no collision can occur when $T_j^{orig} \leq T_i^{dest}$.

Now, assume $T_j^{orig} > T_i^{dest}$. Figure 22 depicts such a situation. If $T_j^{orig} \leq T_i^{orig}$ the vertically hatched region exists and no crane can enter it, by the same argument as in the former case. The same is true for the horizontally hatched region when $T_j^{dest} \leq T_i^{dest}$. The diagonally hatched region is always as wide as the wider of the other two regions, because the lift time and the drop time are both nonnegative. If only one of the regions exits the diagonally hatched region is at as least as wide as that one. By $(l2)$ and $(l3)$ it always holds that the regions are at least as wide as the buffer size (b).

The last case where we have not yet proven the two operations to be internally collision-free is for $T_j^{orig} > T_i^{dest} \wedge T_j^{orig} > T_i^{orig} \wedge T_j^{dest} > T_i^{dest}$. If $T_j^{dest} > T_i^{orig}$ the two operations have no overlap in position and hence they are naturally collision-free. Therefore, we need to consider only the case where $T_j^{dest} \leq T_i^{orig}$. This implies: $T_i^{dest} < T_j^{dest} \leq T_i^{orig} < T_j^{orig}$. The case is depicted in Figure 23. The hatched area is at least as wide as the buffer size, because $(l4)$ implies: $t_j + p_j + m_{T_j^{orig}T_j^{dest}} - (t_i + p_i + m_{T_i^{orig}T_j^{orig}}) \geq b$.

□

Right crane moves first In Table 3 it is shown how to calculate g_{ij}^r. The calculation is analogue to the one, we have just gone through for left crane before right crane. Operation i is now allocated to the right crane and j to the left crane. Again, in case of any conflict between the two operations, operation i is before operation j. All coordinates are just mirrored, which does not affect any of the movement times and hence the calculations are very similar.

Figure 24 illustrates how $(r1)$ is closely related to $(l1)$. The only difference is the precondition, which is mirrored. The proof of correctness is naturally analogues with the previous case.

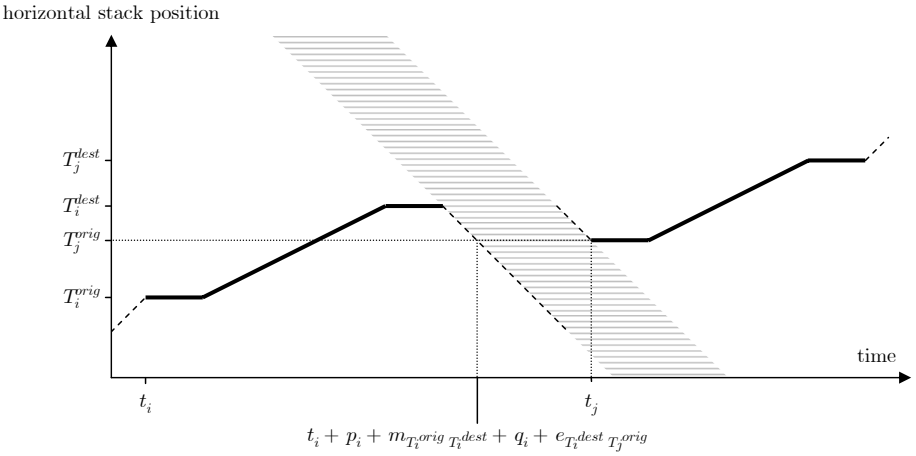

Figure 21: Assume $T_j^{orig} \leq T_i^{dest}$.

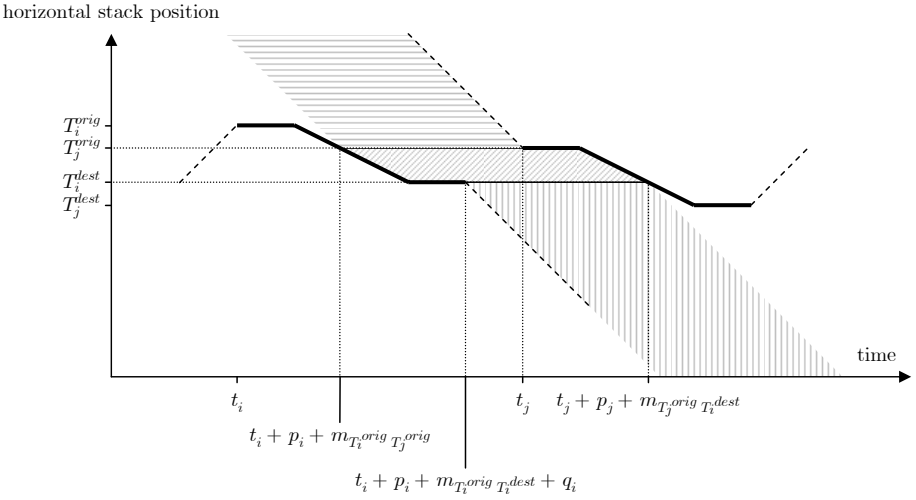

Figure 22: Assume $T_j^{orig} > T_i^{dest}$.

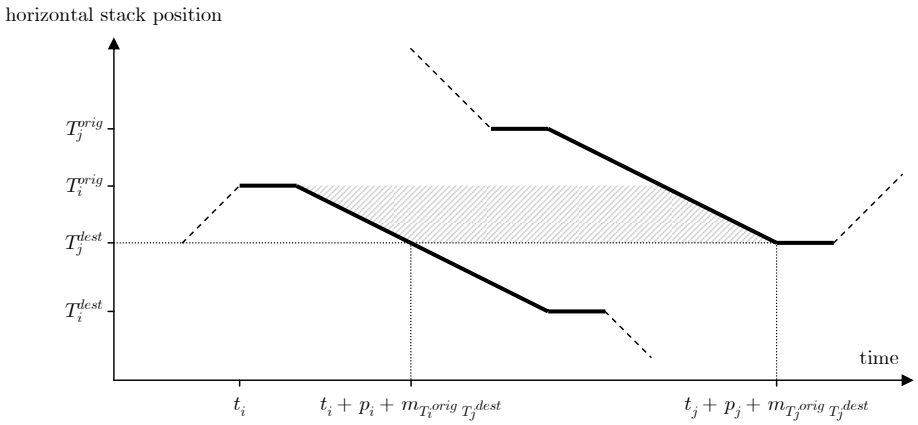

Figure 23: Assume $T_i^{dest} < T_j^{dest} \leq T_i^{orig} < T_j^{orig}$.

	Precondition	Gap
$(r1)$	$T_j^{orig} \geq T_i^{dest}$	$g_{ij}^r = p_i + m_{T_i^{orig} T_i^{dest}} + q_i + e_{T_i^{dest} T_j^{orig}} + b$
$(r2)$	$T_j^{dest} \geq T_i^{dest} > T_j^{orig}$	$g_{ij}^r \geq p_i + m_{T_i^{orig} T_i^{dest}} + q_i + b - (p_j + m_{T_j^{orig} T_i^{dest}})$
$(r3)$	$T_i^{dest} > T_j^{orig} \geq T_i^{orig}$	$g_{ij}^r \geq p_i + m_{T_i^{orig} T_j^{orig}} + b$
$(r4)$	$T_i^{dest} > T_j^{dest} \geq T_i^{orig} > T_j^{orig}$	$g_{ij}^r = p_i + m_{T_i^{orig} T_j^{dest}} + b - (p_j + m_{T_j^{orig} T_j^{dest}})$
$(r5)$	Otherwise	$g_{ij}^r = -\infty$

Table 3: Calculation of the required gap between two operations. Operation i is allocated to the right crane and operation j to the left crane. In conflict, i is moved before j.
Otherwise means: $T_j^{orig} < T_i^{orig} \wedge T_j^{orig} < T_i^{dest} \wedge T_j^{dest} < T_i^{orig} \wedge T_j^{dest} < T_i^{dest}$

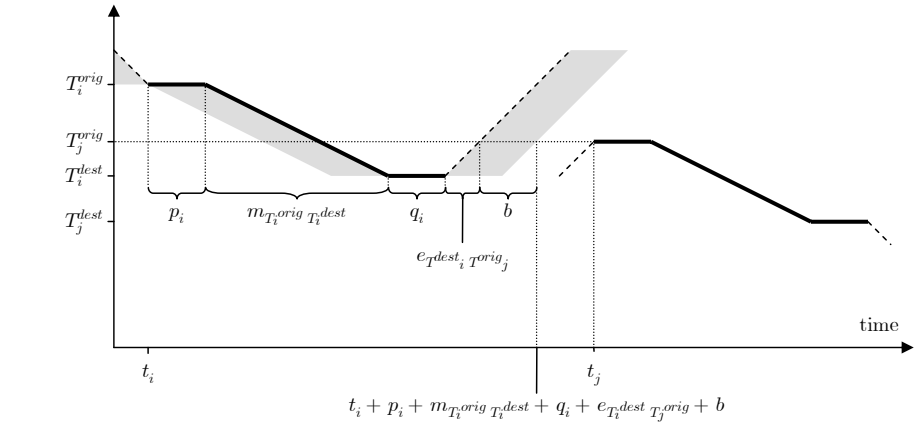

Figure 24: The situation of Figure 16 mirrored vertically. Operation i is now allocated to the right crane.

Example 1 (continued). With these definitions we can illustrate how to calculate the coefficients for the generalized precedence constraints of Example 1. We have the three sets of coefficients: g^s_{ij}, g^l_{ij}, and g^r_{ij} represented by the three matrices of Figure 25. First, we use the precedence constraints of Figure 14 to fill in the ∞ values. This includes the entailed precedence constraints (e.g. $a_1 \to a_2 \wedge a_2 \to a_3 \Rightarrow a_1 \to a_3$). In this example, we have for all operations: $p_i = 1, q_i = 1$, and $b = 1$. $m_{T_xT_y}$ and $e_{T_xT_y}$ are equal and are set to the horizontal distance between the two stacks, cf. Figure 3 (e.g. $m_{T_1T_{exit}} = 4$). Three examples of the calculations for the matrices are shown below ($g^r_{a_1a_2}$ is calculated from $(r2)$+$(r3)$ and $g^r_{a_1a_4}$ is calculated from $(r4)$).

$$
\begin{aligned}
g^s_{a_2a_3} &= p_{a_2} + m_{T^b_{a_2}T^e_{a_2}} + q_{a_2} + e_{T^e_{a_2}T^b_{a_3}} = 1+3+1+1 = 6 \\
g^r_{a_1a_2} &= \max\{p_{a_1} + m_{T^b_{a_1}T^e_{a_1}} + q_{a_1} + b - (p_{a_2} + m_{T^b_{a_2}T^e_{a_1}}), p_{a_1} + m_{T^b_{a_1}T^b_{a_2}} + b\} \\
&= \max\{1+1+1+1-(1+1), 1+0+1\} = 2 \\
g^r_{a_1a_4} &= p_{a_1} + m_{T^b_{a_1}T^e_{a_4}} + b - (p_{a_4} + m_{T^b_{a_4}T^e_{a_4}}) = 1+0+1-(1+2) = -1
\end{aligned}
$$

g^s_{ij}	a_1	a_2	a_3	a_4	a_5
a_1	$-$	4	4	6	6
a_2	∞	$-$	6	10	10
a_3	∞	∞	$-$	8	8
a_4	∞	∞	6	$-$	6
a_5	∞	∞	6	∞	$-$

g^l_{ij}	a_1	a_2	a_3	a_4	a_5
a_1	$-$	5	$-\infty$	7	7
a_2	∞	$-$	7	11	11
a_3	∞	∞	$-$	9	9
a_4	∞	∞	$-\infty$	$-$	7
a_5	∞	∞	$-\infty$	∞	$-$

g^r_{ij}	a_1	a_2	a_3	a_4	a_5
a_1	$-$	2	5	-1	-1
a_2	∞	$-$	4	-1	-1
a_3	∞	∞	$-$	$-\infty$	$-\infty$
a_4	∞	∞	7	$-$	2
a_5	∞	∞	7	∞	$-$

Figure 25: Coeffecients of generalized precedence constraints for Example 1.

5.3 Objective Function

The objective function is, as it was described in Section 3.2, to minimize the maximum tardiness of the schedule. At the same time, a good schedule includes many optional operations. The sum of the priorities of the included optional operations is used to evaluate this criterion. The objective function is two-layered so that minimization of maximum tardiness is always prioritized over the second objective. However, we still require all operations with priority ∞ (compulsory operations) to be in the schedule.

5.4 Generic Formulation of the Crane Scheduling Problem

We are now able to abstracting fully from the real-world context and introduce an explicit formulation of the Crane Scheduling Problem as a parallel scheduling problem with generalized precedence constraints, non-zero release times, and sequence-dependent setup time. Using the three-field notation of Graham et al. [8] extended by Brucker et al. [2] and Allahverdi et al. [1] we denote the problem $R2|temp, r_j, s_{ijm}|T_{max}$.

Sets:

$\mathcal{O}$	The set of operations.
$\mathcal{C} = \{C^l, C^r\}$	The two cranes, left crane and right crane respectively.

Decision variables:

$x_i \in \mathbb{B}$	$i \in \mathcal{O}$	1 if operation i is included in the schedule, 0 otherwise.
$t_i \in \mathbb{Z}$	$i \in \mathcal{O}$	Start time of operation i.
$c_i \in \mathcal{C}$	$i \in \mathcal{O}$	The crane allocation of operation i.
$y_{ij} \in \mathbb{B}$	$i \in \mathcal{O}, j \in \mathcal{O}$	1 if the temporal relation between operation i and operation j must be respected, 0 otherwise.
$\tau_i \in \mathbb{Z}$	$i \in \mathcal{O}$	Tardiness of operation i.

Parameters:

$g^s_{ij} \in \mathbb{Z}$	$i \in \mathcal{O}, j \in \mathcal{O}$	The required gap between operations i and j when allocated to the same crane and i has priority in a conflict.
$g^l_{ij} \in \mathbb{Z}$	$i \in \mathcal{O}, j \in \mathcal{O}$	The required gap between operations i and j when allocated respectively to the left crane and the right crane and i has priority in a conflict.
$g^r_{ij} \in \mathbb{Z}$	$i \in \mathcal{O}, j \in \mathcal{O}$	The required gap between operations i and j when allocated respectively to the right crane and the left crane and i has priority in a conflict.
r_i	$i \in \mathcal{O}$	Release time of operation i.
d_i	$i \in \mathcal{O}$	Due date of operation i.
p_i	$i \in \mathcal{O}$	Priority (weight) of operation i.
t_i^{max}	$i \in \mathcal{O}$	Deadline of operation i.

The Constraint Programming Model:

$$\min \max \tau_a \text{ and secondly } \max \sum_{i \in A} p_i x_i \tag{1}$$

$$\tau_i = \max\{0, t_i - d_i\} \qquad \forall i \in \mathcal{O} \tag{2}$$

$$x_i = 1 \wedge x_j = 1 \Rightarrow y_{ij} = 1 \vee y_{ji} = 1 \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O}, i \neq j \tag{3}$$

$$t_i \geq r_i \qquad \forall i \in \mathcal{O} \tag{4}$$

$$t_i \leq t_i^{max} \qquad \forall i \in \mathcal{O} \tag{5}$$

$$y_{ij} = 1 \wedge c_i = C^l \wedge c_j = C^l \Rightarrow t_i + g_{ij}^s \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{6}$$

$$y_{ij} = 1 \wedge c_i = C^r \wedge c_j = C^r \Rightarrow t_i + g_{ij}^s \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{7}$$

$$y_{ij} = 1 \wedge c_i = C^l \wedge c_j = C^r \Rightarrow t_i + g_{ij}^l \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{8}$$

$$y_{ij} = 1 \wedge c_i = C^r \wedge c_j = C^l \Rightarrow t_i + g_{ij}^r \leq t_j \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{9}$$

$$p_i = \infty \Rightarrow x_i = 1 \qquad \forall i \in \mathcal{O} \tag{10}$$

$$x_i \in \mathbb{B}, t_i \in \mathbb{Z}, c_i \in \mathcal{C}, y_{ij} \in \mathbb{B}, \tau_i \in \mathbb{Z} \qquad \forall i \in \mathcal{O}, \forall j \in \mathcal{O} \tag{11}$$

The model (1)-(11) captures all the problem properties that have been described in this section. (1) is the objective function, which has two criteria. (2) sets the tardiness of each operation. (3) ensures that if both operations are included in the schedule, then their internal precedence constraints must be respected either in one direction or the other. Operations cannot be started before their release date (4) and must be scheduled within the horizon (5). (6)-(9) connect the decision on crane allocation with the correct precedence constraints. Finally, (10) makes sure that all compulsory operations are included in the schedule, and (11) gives the domains of the decision variables.

The parameter p_i is handed down directly from the planning solution. r_i and d_i are calculated as $r_i = ELT_i - dur_i$ and $d_i = ALT_i - dur_i$, where dur_i is the duration of an operation, i.e. $dur_i = p_i + m_{T_i^{orig} T_i^{dest}} + q_i$. g_{ij}^s, g_{ij}^l, and g_{ij}^r are calculated as described in Section 5.2.

A feasible solution to this problem, is a feasible assignment of values to all decision variables. We also use graphical representations of solutions. These are presented in Section 5.5.

Example 1 (continued). We consider again Example 1. We have the 5 operations of Figure 4 which make up the set of operations $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5\}$. We have all temporal relations from Figure 25. The duration of each operation and subsequently the release date and due date of each operation is calculated in the table. In this example the scheduling horizon is $0 \leq t_i \leq 22$:

Operation (i)	dur_i	r_i	d_i	t_i^{max}	p_i
o_1	3	0	19	19	∞
o_2	5	0	5	17	∞
o_3	3	8	9	19	∞
o_4	4	0	18	18	∞
o_5	4	0	18	18	∞

An optimal solution is (Solution 1a):

Operation (i)	x_i	t_i	c_i	y_{io_1}	y_{io_2}	y_{io_3}	y_{io_4}	y_{io_5}	τ_i
o_1	1	0	C^r	–	1	1	1	1	0
o_2	1	2	C^l	0	–	1	1	1	0
o_3	1	9	C^r	0	0	–	1	1	0
o_4	1	12	C^l	0	0	0	–	1	0
o_5	1	18	C^l	0	0	0	0	–	0

Another solution that we will get back to is (Solution 1b):

Operation (i)	x_i	t_i	c_i	y_{io_1}	y_{io_2}	y_{io_3}	y_{io_4}	y_{io_5}	τ_i
o_1	1	0	C^r	–	1	1	1	1	0
o_2	1	4	C^r	0	–	1	1	1	0
o_3	1	10	C^r	0	0	–	1	1	1
o_4	1	3	C^l	0	0	0	–	1	0
o_5	1	9	C^l	0	0	0	0	–	0

5.5 Alternative Representations of Solutions to the Crane Scheduling Problem

5.5.1 Gantt Chart

Solution 1a of Example 1 is vizualized in a Gantt chart in Figure 26. The Gantt chart does not depict the value of neither y_{ij}-variables nor τ_i-variables.

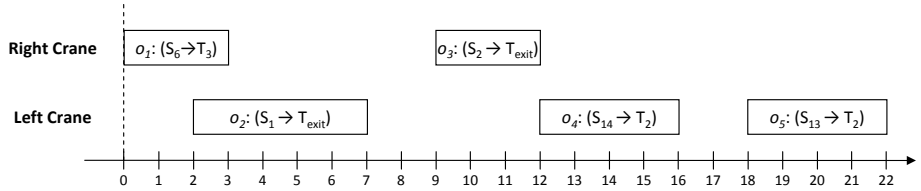

Figure 26: Gantt chart of optimal solution to the scheduling problem of Example 1.

Solution 1b is illustrated in Figure 27. The solution is more compact and may actually look more attractive. However, the due date of o_3 is violated and hence this solution is worse than Solution 1a.

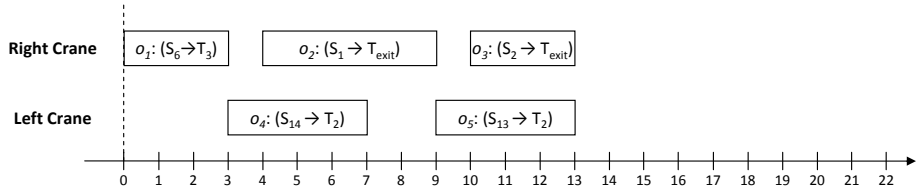

Figure 27: Gantt chart of another solution to the scheduling problem of Example 1.

5.5.2 A List of Operations with Crane Allocations

The nature of the problem makes the transitive closure valid for all choices of priority on conflicting operations, i.e. if operation i is before j (with respect to conflicts) and j is before k then we may assume that i is before k ($y_{ij} = 1 \wedge y_{jk} = 1 \Rightarrow y_{ik} = 1$). We find this propertyalso in lists and hence we may use a list to represent all sequencing decisions. If further we state the crane allocation of each operation, c_i, and if we assume that all operations are scheduled at the earliest possible time according to the given sequence and the crane allocations, then the list representation is sufficient to explicitly represent the solution. The earliest possible times are found in polynomial time by running through the list. For every operation i the generalized precedence constraints to all preceding operations are checked and the most limiting of those determine the starting time of operation i. We adapt the graphical representation from the planning solutions but add information on the crane allocation. Further, we leave out the operations which are not included in the schedule (where $x_i = 0$). We still lack information on t_i and τ_i and therefore the objective function of the solution is not immediately available, but can be calculated by running through the list. The two solutions from before are represented as seen on Figure 28 and Figure 29.

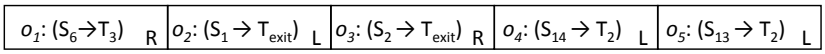

Figure 28: Alternative graphical representation of solution 1a of Example 1.

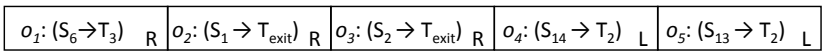

Figure 29: Alternative graphical representation of solution 1b of Example 1.

The advantage of this representation is realized from the two figures (Figure 28 and Figure 29). The only difference between the two solutions is the change in crane allocation of operation o_2. All other variable changes (that were observed on Figure 27) can be interpreted as consequences of this variable change. Another nice feature of the list representation is that any permutation that respects all precedence relations is also feasible with respect to (2)-(4) + (6)-(11). Only the scheduling horizon is possibly violated.

5.5.3 Extended Alternative Graph Formulation of The Crane Scheduling Problem

We now introduce an extended Alternative Graph Formulation of the Crane Scheduling Problem. The Alternative Graph Formulation was first introduced by Mascis and Pacciarelli [17] and is based on a generalization of the disjunctive graph, introduced by Roy and Sussman [20]. In the alternative graph, the longest path through the graph represents a feasible schedule. The graph is a task-on-nodes representation, where each operation is represented by a node and

arcs between nodes represent transitions from one operation to the next. The graph may include alternative arc pairs. From an alternative arc pair, one of the arcs is omitted whereas the other arc stays in the graph. The optimization problem is to select the alternative arcs so that some criterion is optimized. The longest path from an initial node to each node, determines the start time of the operation represented by that node. In our case, the objective hence is to create longest paths that lead to a minimum violation of due dates.

In The Crane Scheduling Problem, we have release dates and due dates of operations. These have to be included in the Alternative Graph Formulation, and hence we get an Alternative Graph Formulation with Time Windows. The objective is to violate the due dates of the nodes as little as possible. We have a number of operations which are optional. To include this property in the graph, we introduce the notion of alternative nodes. These are nodes which may be omitted from the graph at a certain cost (equivalent to receiving a prize for inclusion in the graph). This is the Alternative Graph Formulation with Time Windows and Optional Nodes or just the Extended Alternative Graph Formulation. In Figure 30 the relation between two operations are shown in the alternative graph formulation. The two operations are represented by 4 nodes each (L1, L2, R1, R2). The time windows and node weights are not shown. Alternative arcs are dashed and the alternative arc pairs are visualized by the gray circles connected by dotted lines.

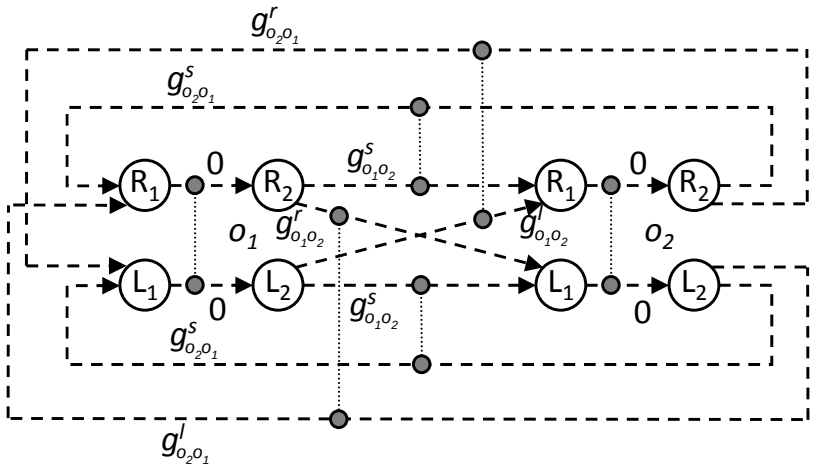

Figure 30: Relation between two operations in the Alternative Graph Formulation.

The Alternative Graph Formulation of Figure 30 has 6 alternative arc pairs and hence 6 decisions to be made. 3 of those are, however, irrelevant if we make the decisions in a certain order. The two internal arc pairs, from R1 to R2 and from L1 to L2 in each node, represent the decision on crane allocation. When we have made a decision on the two pairs, the graph may look like Figure 31.

Only the internal arcs have end points in R2 and L2 of any node, and hence a number of arcs become irrelevant with respect to the longest path. We are able to reduce the graph significantly to the graph of Figure 32.

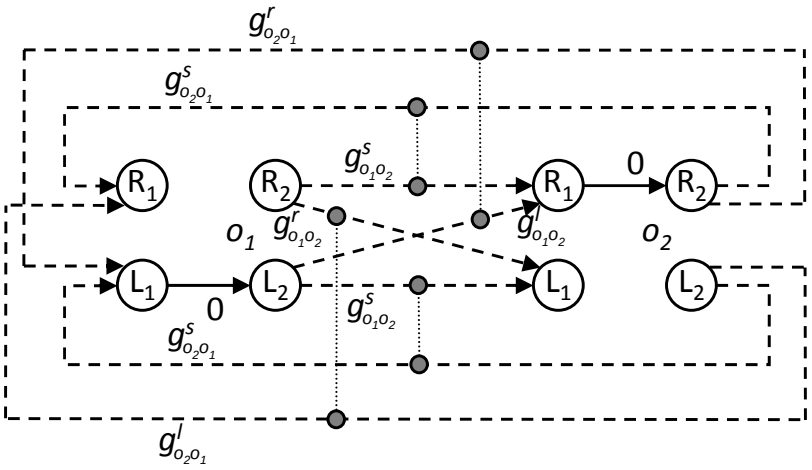

Figure 31: Relation between two operations in the Alternative Graph Formulation with given crane allocation.

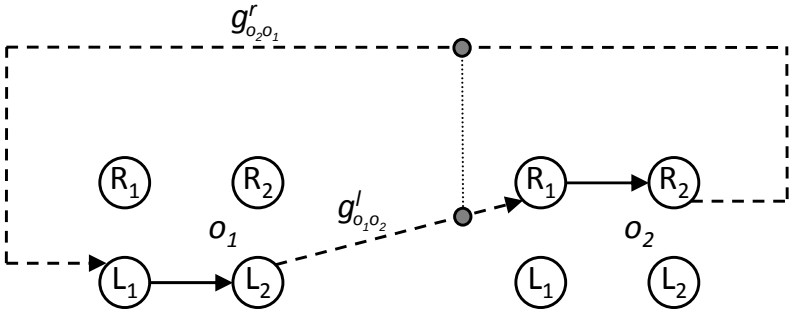

Figure 32: Relation between two operations in the Alternative Graph Formulation with given crane allocation after reduction.

We may also reinterpret the crane allocation decision as being a decision on alternative nodes instead of alternative arcs. Deviating slightly from the Alternative Graph Formulation, however, the idea is the same, and we are now able to illustrate each operation as two alternative nodes and we have the simple graph of Figure 33(a).

When the precedence decision has been made on the two operations, we are left with only one arc as shown in Figure 33(b).

As it is clear from Figure 30, the alternative graph representation of The Crane Scheduling Problem is not appropriate for illustrating examples with more than a few nodes. However, the graph of Figure 33(b) is simple and may be suitable for graphical representation of a solution, as it shows information on the precedence relations. Next, we use this representation for Example 1.

Example 1 (continued). Figure 34 depicts Solution 1a. The longest path from an initial node to all other nodes is highlighted and the earliest possible

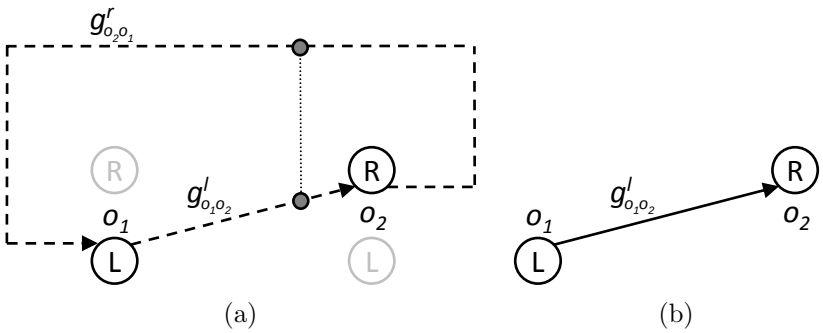

Figure 33: (a) Relation between two operations in the Alternative Graph Formulation with given crane allocation and with alternative node representation. (b) Only one arc is left when all decisions have been made.

starting time for each operation is easily calculated as the length of this path. An arc from the initial node to other nodes may be introduced to respect the release dates. This is illustrated for operation o_3. The nature of the problem ensures that for operations scheduled on the same crane, the operation immediately preceding another operation is always the most restrictive one, and other relations on the same crane may hence be disregarded. This is illustrated in Figure 34 by a dotted line from the node of o_2 to the node of o_5.

	o_1	o_2	o_3	o_4	o_5
$[r_i,d_i]$	[0,19]	[0,5]	[8,9]	[0,18]	[0,18]
p_i	∞	∞	∞	∞	∞
t_i	0	2	9	12	18

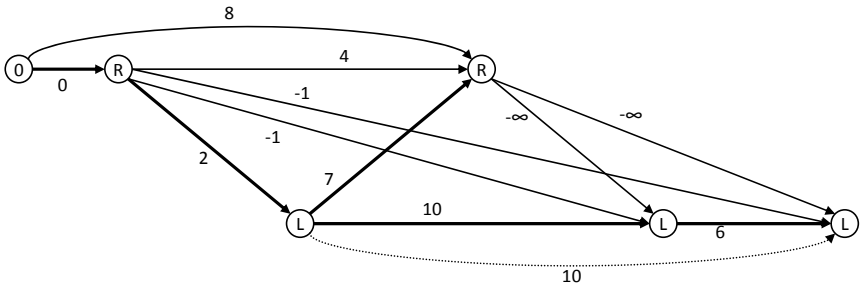

Figure 34: Solution 1a depicted by a reduced Alternative Graph.

Likewise we depict Solution 1b in Figure 35.

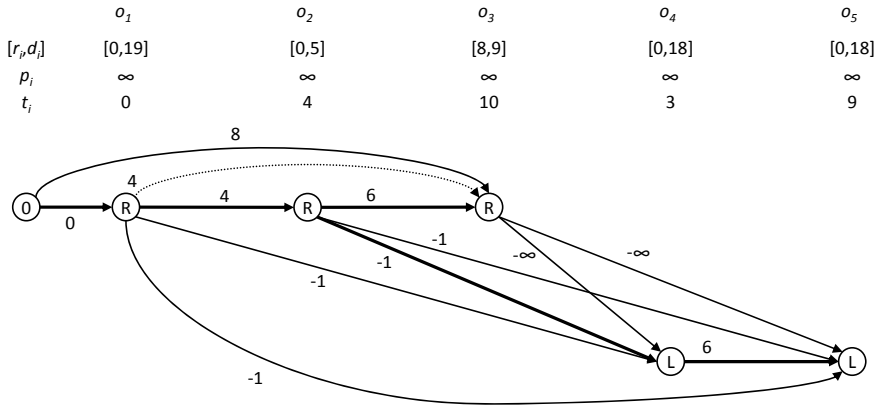

Figure 35: Solution 1b depicted by a reduced Alternative Graph.

5.5.4 Time-Way Diagrams

Yet another way of visualizing a solution graphically is by using Time-Way Diagrams as the ones we used in Section 5.2. Solid lines indicate that the crane is processing an operation, whereas dashed lines indicate that the crane is either waiting or moving to the start position of the next operation. Solution 1a and Solution 1b are depicted in Figure 36 and Figure 37, respectively.

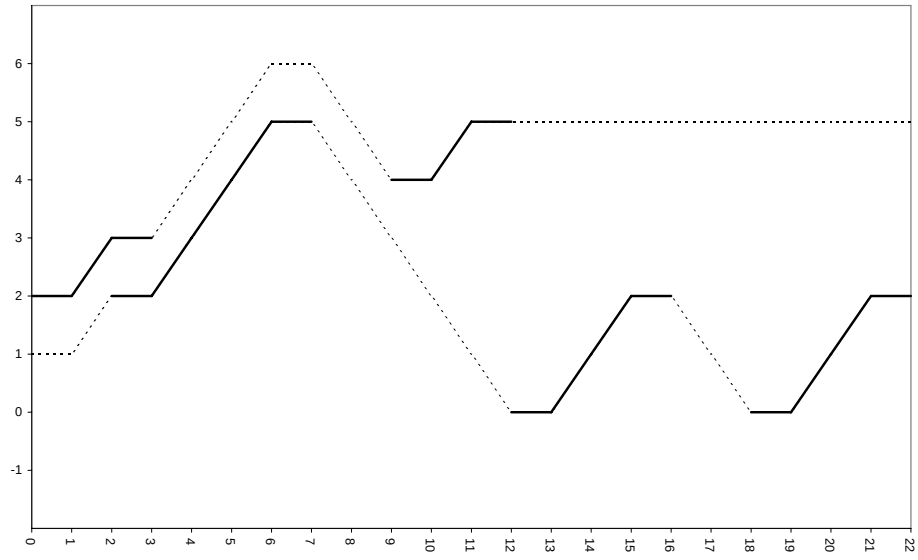

Figure 36: Time-Way Diagram of Solution 1a.

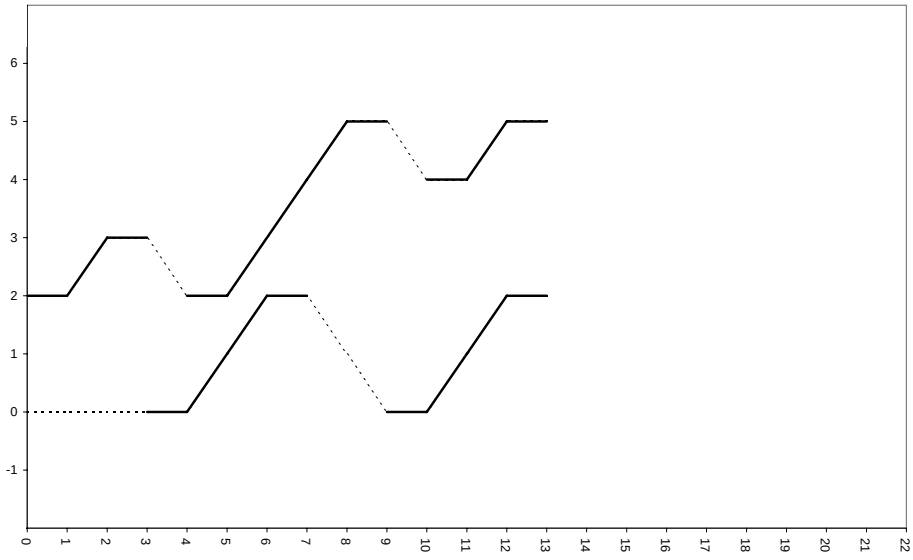

Figure 37: Time-Way Diagram of Solution 1b.

6 Alternative formulations

The proposed model is just one out of several promising ways of modeling the problem at hand. In the following, we describe a number of variations of the model, along with the pros and cons of each.

6.1 Two-stage planning

We may split the planning phase into two stages. Again, we want to separate the important parts of decision making, so that decision can be made with the correct abstraction level. Here the idea is to split the problem into:

1. A decision on end position of a slabs, regardless of the means of getting them there.

2. Creating a planning solution with all necessary specifications, i.e. explicitly list all operations.

The first part is an abstraction of the planning problem described in Section 4. We present a MIP-model of the problem. In the model, we determine the end stack of each slab. For most slabs, this is naturally the same as the initial stack for the particular scheduling horizon. In the model, we do not include probabilities in the calculation of false positions, but we use the simplification, where a slab is in a false position, if it is on top of another slab with earlier due date.

Sets:

$i \in I$	Stacks
$j \in J$	Slabs
$J^{due} \subseteq J$	Slabs with a due date in the current scheduling horizon

Parameters:

c_{ij}	=	estimated cost of moving slab j to stack i.
$ontopof(j, j')$	=	$\begin{cases} 1 & \text{Slab } j \text{ is on top of slab } j' \text{ and must hence be moved if } j' \text{ is to be mov} \\ 0 & \text{Otherwise} \end{cases}$
i_j^0	=	Initial stack of slab j.
$j_{ij}^{reshuffle}$	=	The lowest slab in stack i with a due date before slab j.
α	=	Cost of one false position.
d	=	Maximum height of the stacks.

Variables:

x_{ij}	=	$\begin{cases} 1 & \text{Stack } i \text{ contains slab } j \text{ by the end of the day} \\ 0 & \text{Otherwise} \end{cases}$
m_j	=	$\begin{cases} 1 & \text{Slab } j \text{ is moved} \\ 0 & \text{Otherwise} \end{cases}$
$\hat{g}_j$	=	$\begin{cases} 1 & \text{A slab with an earlier due date is found below slab } j \text{ by the end of the day} \\ 0 & \text{Otherwise} \end{cases}$

The model:

$$\min \sum_j \sum_i c_{ji} x_{ji} + \alpha \sum_j \hat{g}_j \tag{12}$$

$$\sum_i x_{ji} = 1 \qquad \forall j \tag{13}$$

$$\sum_j x_{ji} \leq d \qquad \forall i \tag{14}$$

$$m_j = 1 - x_{ji_j^0} \qquad \forall j \tag{15}$$

$$m_j \geq m_{j'} \qquad \forall j, j' | ontopof(j, j') \tag{16}$$

$$\hat{g}_j + m_{j_{ij}^{reshuffle}} \geq x_{ji} \qquad \forall j, i | j_{ij}^{reshuffle} \neq \emptyset \tag{17}$$

$$x_{ji^{exit}} = 1 \qquad \forall j \in J^{due} \tag{18}$$

$$x_{ji^{train}} = 0 \qquad \forall j \tag{19}$$

The objective function (12) contains two terms. The first terms is related to the cost of moving a slab to a stack. Typically, $c_{ji_j^0} = 0$, so it does not cost anything to leaving the slab in its current stack. We have the following constraints. All slabs must have exactly one end stack (13). There is a maximum height for a stack (14). If a slab leaves its original stack it counts as a move and the m_j-variable is updated accordingly (15). If a slab is moved, all slabs on top must be moved as well (16). For most end positions of a slab, we have to check if the slab ends up in a false position (17). Either the slab is in a false position ($\hat{g}_j = 1$) or all slabs that could cause a false position have been moved from that stack ($m_{j_{ij}^{reshuffle}} = 1$). Finally, we fix some of the decision variables. All due slabs must be moved to the roller table (18) and incoming slabs must be moved to the yard (19).

The model presented is a slight relaxation of what was described above. When a slab is moved, we do not keep track of its depth in the new stack, and hence we lose information on the internal ordering in the new stack. Modeling this information has also been tried, and it introduces a large number of new decision variables which makes the model less suitable for practical applications. The model can be solved by standard MIP-solvers, but as the practical problems are usually very large, metaheuristics may be better suited for the purpose.

The planning part has two stages, meaning that from a solution to (12)-(19) we need to create a planning solution with the same information level as was described in Section 4. For this we use knowledge from the artificial intelligence field, where a well known planning problem is found in the Blocks World (e.g. [23]). In Blocks World, an initial state and a goal state is given and the problem is to find the minimum number of stack operations (moves) that takes us from the initial to the goal state. In Blocks World, we are allowed to lift only the topmost block of a stack and moved blocks are always put on top of their new stack. This general problem resembles very well what we need in the second part of the planning process.

6.2 Planning-scheduling feedback

Another possibility that was considered in the modeling process, was to create a two-stage method with feedback between the two parts. The idea is to use

knowledge from the schedules when solving the planning problem. E.g., after generating an initial planning solution, a schedule is created with some apparent bottlenecks. This may be repaired in a new plan, by altering certain decisions. We may run several planning-scheduling iterations and hopefully end up with planning solutions that facilitate the generation of a good schedule.

A special case of the planning-scheduling feedback idea is to aim for a very large number of feedback iterations by e.g. not including priorities for optional operations. We may include only a subset of the optional operations in the planning solution and by making the optional operations mandatory, we lower the required solution time in the scheduler. We may even fix the sequence of moves completely, i.e. fix all y_{ij} variables of the formulation (1)-(9). The scheduler has much less flexibility and consequently it gets harder to find good solutions. On the other hand, the solution process is very fast, and we may run thousands of iterations, where the planning module is responsible for all the important decisions and the scheduling module is reduced to an evaluation module, where the proposed solution from the planning module is assessed.

6.3 Scheduling: Allow slight planning modifications

Yet another possible extension to the method is to allow slight planning modifications in the scheduler. There may be some operations, which are particularly difficult to schedule, and since we actually generated the operations ourselves in the planning module, we are also allowed to change them, as long as the final plan is not impaired significantly.

7 Solution Method

A solution method is implemented based on the presented model. In the following, we present two greedy methods, one for the planning problem and one for the scheduling problem. The two methods are straight-forward in their implementation and more sophisticated methods will evidently enhance performance. The simple methods are still able to generate results, and hence we will use them to assess the value of the model.

7.1 Planning

The planning method will give as output a solution as described in Section 4. When the final schedule is created in the second stage of the method, all precedence constraints are respected, and hence the sequence of operations that we specify in the planning solution fully determines the state of the yard. As a result, we are able to update the yard state, as the operations are added to the solution. For any partial solution, we therefore have a current yard state. When we refer to location of slabs, it is with respect to the current state of the yard.

The implemented method is divided into three steps:

- Generation of operations for slabs that must leave the yard during the scheduling horizon (*exit-slabs*).
- Generation of operations for incoming slabs (*arrival-slabs*).
- Generation of other operations.

In this solution method, we treat the three steps separately, one by one.

7.1.1 Leaving slabs

First, we generate a list of operations for the slabs that must leave the yard during the scheduling horizon. We already have their Aim Leave Time and hence we have a predetermined ordering of these operations. The slabs may not be on top of their stacks and hence we may need to generate *reshuffle* operations also, where the slabs on top are moved to other stacks.

For reshuffle operations we must specify a destination stack. The destination stack is chosen from a number of criteria. First, we disallow movement to stacks still containing exit-slabs. Moving a slab to such a stack will trigger another reshuffle-operation later, where the same slab has to be reshuffle again. This should be avoided if possible. Further, when choosing a destination stack, we look for stacks within a short range. This limits the duration of the operation and at the same time decreases the risk of crane collision involving this operation. We also look for stacks where the slab has a small chance of being in a new false position (and hence in need of another reshuffle in a future plan).

7.1.2 Incoming slabs

When all exit-operations have been generated, we proceed with the arrival-operations. For each slab on the railway cars, we generate an operation that will bring the slab to the yard. As each slab leaves the yard along with the rest

of the slabs of its batch, we try to group the slabs by batch id in the yard. When choosing a destination for these slabs we also use the considerations listed for reshuffle-operations in the previous section. It is particularly important to keep the sum of false position probabilities low.

All arrival-operations are sequenced after the exit-operations. This does not necessarily mean that they are also scheduled later than all arrival-operations. The reason why the operations are sequence in this way, in the planning solution, is that all stacks involved in both exit- and arrival-operations, will have the exit-operation executing first, which is obviously a desirable feature. To introduce flexibility in the scheduler, we try to select destination stacks that do not have any outgoing exit-operations.

The order of arrival-operations is partially predetermined. We have to move the slabs from top to bottom from the stacks of the railway cars. We do, however, have a choice between the stacks in the Railway Cars.

7.1.3 Other operations

Last, we generate a number of operations that are not mandatory for feasibility of schedules, but that will increase the quality of the solution by reducing z_2 and z_3 (described in Section 4.3), i.e. the operations will reduce the total false position probability and may also move slabs with an upcoming due date closer to the exit stack. These operations are not necessarily sequenced last in the planning solution. As the operations are generated, different positions in the sequence of operations are considered, and the most suitable position is chosen.

Example 1 (continued). To illustrate the effect of each of the three steps, we again consider Example 1, and show how Solution 3 is generated. The three steps and the (partial) planning solution for each step is depicted in Figure 38.

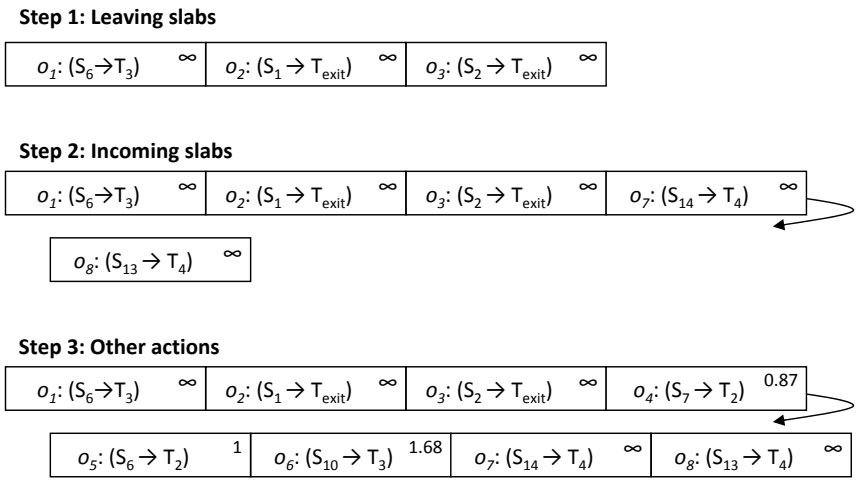

Figure 38: Generation of Solution 3 of Example 1: The three steps and the (partial) planning solution for each step.

7.2 Scheduling

Given a planning solution, we need to schedule the operations on the two cranes. The generic formulation is given in Section 5.4. In the following we describe a greedy heuristic that the current implemented is based on.

The heuristic is very simple. We process the operations in the order given in the planning solution. For each operation, the earliest possible time of initiation is calculated for both cranes. The operation is allocated to the crane that is able to initiate first. As we have release times for operations, there may still be some waiting time from the preceding operation to the current one. Therefore, we check if we are able to squeeze in any of the unscheduled operations. The operations with high priority are preferred over the others. When squeezing in operations like this, we need to make sure that all precedence constraints are respected.

7.2.1 Variations

The proposed heuristic is very simple and has some apparent flaws. The most apparent issue is that it is a greedy heuristic and therefore has no "foresight", so the operations may be distributed evenly on the two cranes in the beginning of the schedule but in the end, all operations tend to involve the same part of the yard, and hence only one of the cranes is used efficiently. The issue is visible in the time-way diagram of Figure 40.

This issue could be addressed by the implementation of a local search procedure, to enhance the results of a greedy construction heuristic. As a starting point, a steepest descent algorithm would probably increase quality significantly. Adding metaheuristic features to such a search is likely to enhance the solution even further. Preliminary test results from a metaheuristic show promising results.

Another possibility is to use standardized scheduling tools. One such technique uses the model directly in a Constraint Programming solver. The quality of solution from such an approach should also be evaluated.

8 Test results

In the following, we present the preliminary test results. The results are based on a number of assumptions and on simulated data. First, we present a setup for simulating manual planning as it currently conducted in practice. By comparing the solutions of the method presented in this paper to the solutions of such a simulation, we are able to assess the value of the proposed method.

8.1 Simulation of Manual Behavior

To evaluate the quality of the solutions, for each instance, we generate a reference solution that represents the solution obtainable by manual planning. We try to imitate the behavior of the cranes when they are under the control of the individual crane operator.

We model the manual behavior as follows.

- Exit-slabs are dealt with as we approach their deadline. We only have knowledge of the next slab leaving the yard. When that slab is on the roller table, we consider how to move the next slab in the exit sequence.
- If slabs are on top of the exit slabs, we deal with it as it happens, i.e. we do not predict this earlier during the day.
- If a crane has free time in between moves, it will use the time to move slabs from the train wagons to the yard.
- When moving slabs to the yard or when moving slabs around in the yard, we equip the crane operators with a two hour no-block foresight, i.e. they will not choose a stack with slabs that have deadlines within the following two hours.
- In the schedules described earlier, we needed a buffer between cranes, to make the schedules more robust. The buffer is disregarded in this part, as we are not really creating a schedule, rather are we simulating manual planning/scheduling, and hence the operations are to be interpreted as happening in real time and not as a pre-made schedule to be followed.

8.2 Overview: Structured Test

In the following we run a number of simulations. The average yard throughput is fixed in each of the test instances. The throughput is increased in the hard test instances, to check the effect on the quality of the schedules. The simulations are kept as close as possible to the real world conditions. We need to assume / decide a number of things for the simulation. The most important simulation settings are listed below.

- We assume that the requested throughput of the yard for each day is randomly drawn from a Gaussian distribution.
- In the same way, we assume that the production time for each slab is drawn from a Gaussian distribution. The mean of this distribution matches the throughput, so that the production actually calls for the correct number of slabs on average.

	Two-Stage				Manual			
Slabs per day	Avg moves per slab	Avg move duration	Avg deadline violation	Max deadline violation	Avg moves per slab	Avg move duration	Avg deadline violation	Max deadline violation
400	2.37	43.91	0.01	1.66	2.61	46.55	0.24	11.78
450	2.39	43.76	0.02	1.35	2.60	46.28	0.09	8.52
500	2.37	43.84	0.02	1.86	2.61	46.68	5.53	61.13
550	2.38	43.80	0.02	1.86	2.59	46.50	2.94	74.64
600	2.41	43.84	1.08	10.54	2.61	46.62	26.69	234.82
650	2.39	43.91	0.40	7.28	2.58	46.63	27.30	253.55
700	2.38	43.91	0.38	7.51	2.59	46.60	139.64	543.17
750	2.38	43.90	0.72	16.50	2.68	46.61	878.07	1883.29
800	2.39	44.00	2.59	44.61	2.90	46.80	3092.32	4840.96
850	2.40	43.92	1.68	43.33	3.14	46.75	5202.69	7874.27
900	2.41	43.88	13.72	135.50	3.30	46.45	7930.29	13108.21
950	2.39	43.95	47.02	270.68	3.34	46.04	9204.57	15641.77
1000	2.44	43.93	118.06	490.39	3.43	45.72	10704.42	18626.53

Table 4: Comparison of test results from the Two-Stage method and simulation of manual planning. Each value is an average over 10 identical runs. Deadline violations and durations are measured in seconds.

- Furthermore, we also have a through time for a slab, which is the amount of time from arrival to the yard to the due date of that slab. We assume that also the through time is drawn from a Gaussian distribution. The through time directly influences the number of slabs in the yard.
- Finally, batch size and number of slabs on a train are also assumed to be drawn from Gaussian distributions.

We have the following simulation parameters and their corresponding values in our simulation:

Parameter	Mean (μ)	Std dev (σ)
$SlabsPerDay$	Set in test	$\frac{\mu}{10}$
$ThroughTime$	$\mu(\frac{2000}{SlabsPerDay}) \cdot (24 \cdot 60 \cdot 60)^{-1}$	$\frac{\mu}{5}$
$BatchSize$	20	5
$SlabsOnTrain$	200	20
$Buffer$	120	-

In the following table we summarize the results for the preliminary test. Each value is an average over 10 identical runs.

From Table 4 it is clear that the proposed method provides significantly better results than the simulation of manual planning. In the table we have shown four performance measures. For each method, the first column gives the average number of times a slab is moved before it leaves the yard. As slabs in our setup are never transferred directly from train wagons to the exit belt, the minimum number of moves of each slab is 2. This measure illustrates how well the moves are planned, i.e. a low number indicates that the slabs are seldom in the way of others. From the tests, we see a significant difference between the two

methods, especially for the harder problems, where the Two-Stage algorithm on average uses approximately one move less per slab. Also the duration of each move is of interest and is shown in the second column. The difference between the two methods is not remarkable, even though the Two-Stage algorithm is a few seconds faster in all cases. The duration seems stable over the set of test instances.

The two last columns report on deadline violations. The rightmost of the columns gives the maximum deadline violation, which is, as stated earlier, the main objective considered in this work. The first of the two columns reports on the average violations. This is interesting if we assume that the following production is able to catch up on the delays we may have caused. A low average deadline violation is equivalent to a low sum of violations, which is another objective often used for scheduling problems in the literature. For both objectives, we see that the Two-Stage algorithm clearly outperforms the other. The manual planning has severe problems in the hard instances, where the results of the Two-Stage method are still satisfactory. The figures for manual simulation may seem very large, but it is noted that the numbers should only be used for comparison with other similar tests. As soon as a method is unable to keep up with the rate at which slabs enter the yard, it will lead to larger and larger violations as we let the simulation run. In each run of these tests, the two methods naturally span over the same production plan.

Both methods are able to produce results in less than a second. Such computation times are insignificant in these settings and are therefore not compared here.

8.3 Details and comments

The test results of Section 8.2 call for a detailed analysis of the hard instances and the corresponding schedules. A significant advantage of a structured planning approach is that we are able to identify weaknesses and repair the causes of those.

In Figure 39 and Figure 40 we show two Time/Way diagrams for the same planning problem. The schedule of Figure 39 is the result of a simulation of manual behavior, whereas Figure 40 depicts the schedule of the Two-Stage algorithm.

From Figure 39 and Figure 40 we see a significant difference between the two methods. The scheduling algorithm of the Two-Stage approach is able to look far ahead in the schedule and the effect is visible in Figure 40. The schedule is clearly separated in two parts, where each crane is able to handle operations in its own part of the yard. Notice that the exit stack is in horizontal position 7 and hence numerous operations end in that particular column.

To evaluate the quality of the schedules, we may also inspect how other problem values vary over time. In Figure 41 we illustrate the change of different values over time for the schedule of Figure 39. In the two charts from the top, all exit moves are included. In the topmost chart, the time of initiation of each exit move. Each such operation has a time window which is also depicted in the plot. The second chart depicts the same information in a slightly different way. The chart shows the time slack of each operation, i.e. the time distance from the scheduled initiation of an operation to the latest point in time, where delays are avoided. Finally, the third chart illustrates, over time, the remaining

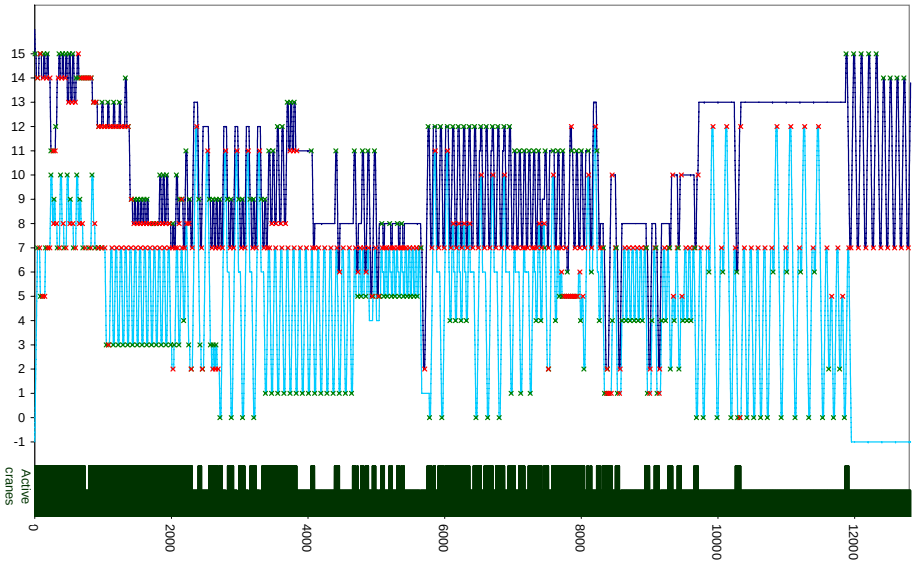

Figure 39: Schedule created by simulation of manual behavior.

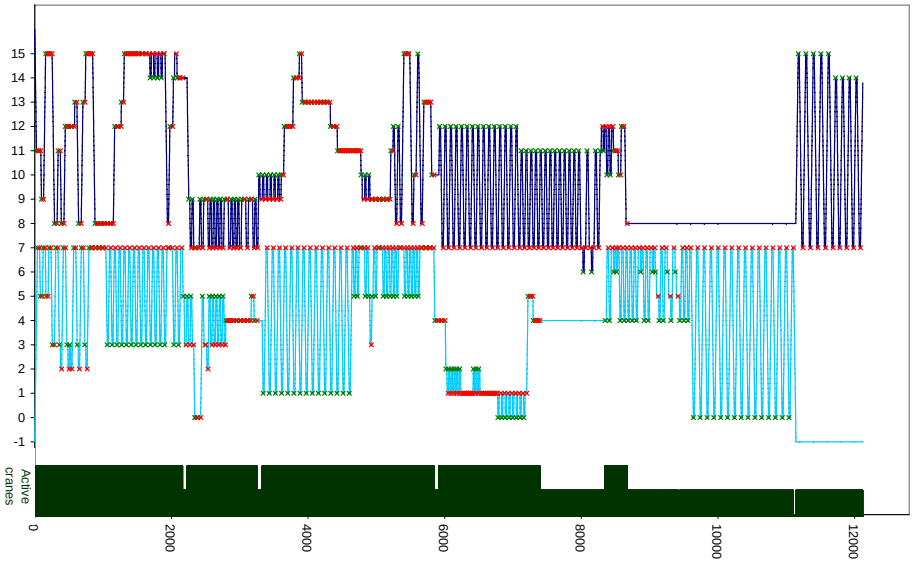

Figure 40: Schedule created by the Two-Stage algorithm.

number of moves of different kinds. The exit moves are the same as depicted in the two other charts. The moves referred to as "Other Moves" are moves which either move slabs which are in the way, or moves which are carried out in order to enhance the state of yard. Figure 42 is the corresponding figure for

the schedule of Figure 40.

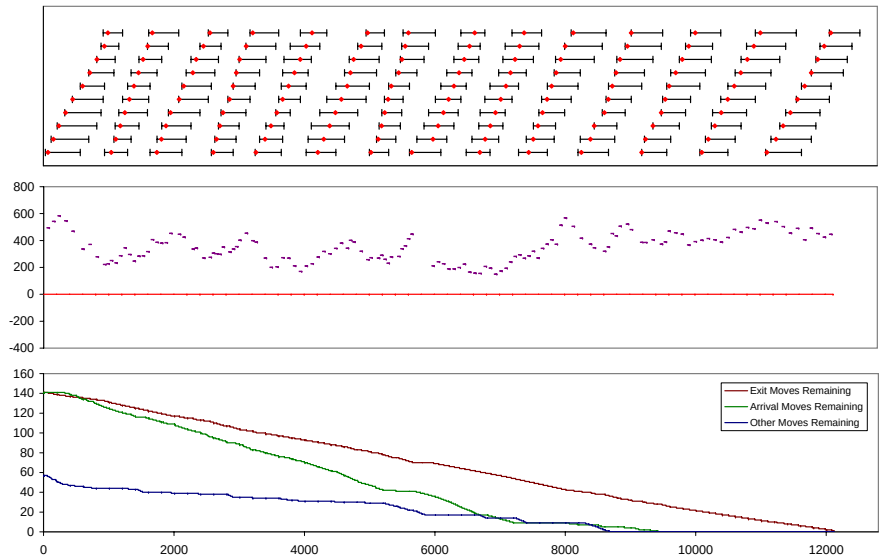

Figure 41: Assessment of schedule by illustration of various problem values over time for schedule of Figure 39.

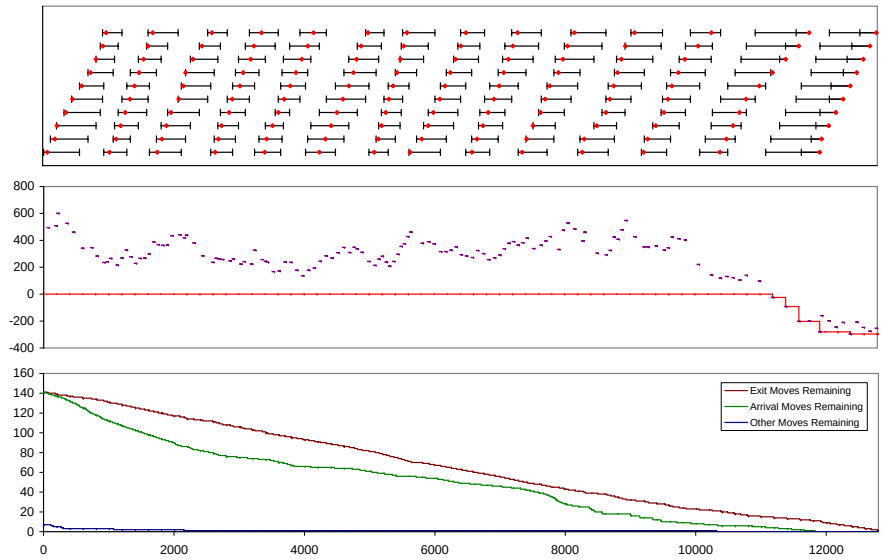

Figure 42: Assessment of schedule by illustration of various problem values over time for schedule of Figure 40.

9 Conclusions

The Slab Yard Planning and Crane Scheduling Problem has been modeled in a novel way that facilitates a beneficial and at the same time transparent optimization. The model is generic enough to capture several variations of the problem. The solution methods adapt to variations of the problem, correspondingly.

From the test results it is clear that the model facilitates an algorithm that is capable of providing solutions, which are superior to the ones achievable by manual planning. The tests are, however, preliminary and based on simulations which rely on a number of assumptions.

The model in itself is a valuable tool for analyzing proposed schedules in a number of various ways. In short term planning, several graphical representations allow for a thorough quality assessment. In long term planning, several optimization criteria have been introduced, and these criteria can be evaluated also for schedules which have been created by other means than the proposed methods.

9.1 Pros and cons of the chosen model

In the paper we have introduced a model that, by splitting The Slab Yard Planning and Crane Scheduling Problem in two stages, facilitate a solution procedure that is clear in the formulation of objectives and is able to generate superior schedules by targeting the problem in two different abstraction levels. We will shortly discuss the conclusions that we are able to draw on the value of the model, based on the findings presented in this paper.

The chosen model needs no feedback loop between the planning and scheduling procedures. This means that we are able to find solutions to problems of a realistic size in less than a second. A strict time limit may be a part of the requirement of a potential user, and hence it is important that we are able to comply with such requirements. Another advantage of not having a feedback loop is that the scheduler has time enough to find near-optimal solutions. This comes at a cost, however. The schedules are near-optimal with respect to the given planning solution. The planning solution has not been adapted to any of the challenges discovered in the scheduling stage. Planning decisions are made on a high abstraction level and hence they are, hopefully, suitable for high quality in long term planning. In short term planning, the plan may have some shortcomings. The scheduler will circumvent such shortcomings as much as possible. It may be advantageous to allow the scheduler to make small alternations to the plan, in order to optimize the short term planning further.

9.2 Future work

Future work should be aimed at real-world applications. So far, the experimental conclusions are based on simulations and artificially generated data. The true value of this work is in the possible applications. It will be interesting to see how well real problem data fits the assumptions that were made in the generation of artificial problem instances. In a practical application it is possible to tailor the algorithms to fit the exact properties of that particular problem. In this paper, we have made sure not to take advantage of structures in the problem data, as such structures may not transfer to variations of the problem. Therefore, in a

practical application, it may be possible to utilize problem specific knowledge in the creation of the planning and the scheduling method, and get even better results.

References

[1] A. Allahverdi, C.T. Ng, T.C.E. Cheng, and M.Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[2] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[3] A. Che and C. Chu. Single-track multi-hoist scheduling problem: a collision-free resolution based on a branch-and-bound approach. *International Journal of Production Research*, 42(12):2435–2456, 2004.

[4] R. Dekker, P. Voogd, and E. Asperen. Advanced methods for container stacking. *OR Spectrum - Quantitative Approaches in Management*, 28(4):563, 2006.

[5] Y. Dinitz and S. Solomon. Optimal algorithms for tower of hanoi problems with relaxed placement rules. *Lecture Notes in Computer Science*, 4288:36, 2006.

[6] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–93, 1978.

[7] L.M. Gambardella, M. Mastrolilli, A.E. Rizzoli, and M. Zaffalon. An optimization methodology for intermodal terminal management. *Journal of Intelligent Manufacturing*, 12(5-6):521, 2001.

[8] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Discrete Optimisation*, 5:287–326, 1979.

[9] J. Hansen. *Industrialised application of combinatorial optimization.* PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2003.

[10] K.H. Kim and J.W. Bae. Re-marshaling export containers in port container terminals. *Computers and Industrial Engineering*, 35(3-4):655–658, 1998.

[11] K.H. Kim, Y.M. Park, and K.-R. Ryu. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124(1):89–101, 2000.

[12] F.G. König, M. Lübbecke, R. Möhring, G. Schäfer, and I. Spenke. Solutions to real-world instances of pspace-complete stacking. *Algorithms - ESA 2007. Proceedings 15th European Symposium. (Lecture Notes in Computer Science vol. 4698)*, pages 729–40, 2007.

[13] J. Lamothe, C. Thierry, and J. Delmas. A multihoist model for the real time hoist scheduling problem. *Symposium on Discrete Events and Manufacturing Systems. CESA'96 IMACS Multiconference. Computational Engineering in Systems Applications*, pages 461–6, 1996.

[14] J. Leung and G. Zhang. Optimal cyclic scheduling for printed circuit board production lines with multiple hoists and general processing sequence. *IEEE Transactions on Robotics and Automation*, 19(3):480–484, 2003.

[15] J.M.Y. Leung, G. Zhang, X. Yang, R. Mak, and K. Lam. Optimal cyclic multi-hoist scheduling: A mixed integer programming approach. *Operations Research*, 52(6):965–976, 2004.

[16] J. Liu and Y. Jiang. An efficient optimal solution to the two-hoist no-wait cyclic scheduling problem. *Operations Research*, 53(2):313–27, 2005.

[17] A. Mascis and D. Pacciarelli. Machine scheduling via alternative graphs. Report DIA-46-2000, Dipartimento di Informatica e Automazione, Universit a Roma Tre, Roma, Italy, 2000.

[18] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[19] R. Rodosek and M. Wallace. Submitted papers - a generic model and hybrid algorithm for hoist scheduling problems. *Lecture Notes in Computer Science*, 1520:385–399, 1998.

[20] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Note d.s. no. 9 bis, SEMA, 1964.

[21] K.A. Singh, Srinivas, and M.K. Tiwari. Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved parallel genetic algorithms. *International Journal of Production Economics*, 91(2):135–147, 2004.

[22] J. Skov. Scheduling of an anodizing plant at bang & olufsen. Master's thesis, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, August 2007.

[23] J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[24] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26(1):3–49, 2004.

[25] T.C. Sun, K.K. Lai, K. Lam, and K.P. So. Study of heuristics for bidirectional multi-hoist production scheduling systems. *International Journal of Production Economics*, 33(1):207–214, 1994.

[26] L. Tang, J. Liu, A. Rong, and Z. Yang. An effective heuristic algorithm to minimise stack shuffles in selecting steel slabs from the slab yard for heating and rolling. *Journal of the Operational Research Society*, 52(10):1091–1097, 2001.

[27] L. Tang, J. Liu, A. Rong, and Z. Yang. Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules. *International Journal of Production Research*, 40(7):1583–95, 2002.

[28] C. Varnier, A. Bachelu, and P. Baptiste. Resolution of the cyclic multi-hoists scheduling problem with overlapping partitions. *INFOR*, 35(4):309–324, 1997.

[29] M. Zaffalon, A.E. Rizzoli, L.M. Gambardella, and M. Mastroiilli. Resource allocation and scheduling of operations in an intermodal terminal. *Simulation Technology: Science and Art. 10th European Simulation Symposium 1998. ESS'98*, pages 520–7, 1998.

[30] X. Zhu and W.E. Wilhelm. Scheduling and lot sizing with sequence-dependent setup: a literature review. *IIE Transactions*, 38(11):987–1007, 2006.