# A General Framework for Scheduling Equipment and Manpower at Container Terminals

Sönke Hartmann[a,b]

[a]OR Consulting, Bornstraße 6, D-20146 Hamburg, Germany.
E-mail: soenke.hartmann@gmx.de

[b]Christian-Albrechts-Universität zu Kiel, Lehrstuhl für Produktion und Logistik, D-24098 Kiel, Germany. E-mail: hartmann@bwl.uni-kiel.de

**Abstract.** In this paper, we propose a general model for various scheduling problems that occur in container terminal logistics. The scheduling model consists of the assignment of jobs to resources and the temporal arrangement of the jobs subject to precedence constraints and sequence-dependent setup times. We demonstrate how the model can be applied to solve several different real-world problems from container terminals in the port of Hamburg (Germany). We consider scheduling problems for straddle carriers, automated guided vehicles (AGVs), stacking cranes, and workers who handle reefer containers. Subsequently, we discuss priority rule based heuristics as well as a genetic algorithm for the general model. Based on a tailored generator for experimental data, we examine the performance of the heuristics in a computational study. We obtain promising results that suggest that the genetic algorithm is well suited for application in practice.

**Keywords.** Container logistics, container terminal, optimization, scheduling, heuristics, genetic algorithm.

## 1 Introduction

In the 1960s, the container was introduced as a universal carrier for various goods. It soon became a standard in worldwide transportation. The success of the container is associated with the increasing containerization (which means that the number of goods transported in containers has steadily grown) and with increasing world trade. Container terminals are continuously facing the challenge of strong competition between ports and of turning around more and larger ships in shorter times. This leads to the necessity to use the highly expensive terminal resources such as quai cranes, straddle carriers, automated guided vehicles, and stacking cranes as efficiently as possible. A key factor of success is the optimization of the logistic processes. Therefore, many researchers and practitioners have developed optimization approaches for container terminal logistics.

Important optimization problems include the assignment of berths to arriving vessels (see Guan and Cheung [12], Imai et al. [18, 19], Lim [29]) and the assignment of quai cranes to vessels or ship-bays (see Daganzo [4], Peterkofsky and Daganzo [34]). A berth assignment approach which simultaneously considers quai crane capacities has been developed by Park and Kim [33]. Scheduling the container transport on the terminal has been studied for two different types of equipment, namely straddle carriers (see Böse et al. [3], Kim and Kim [26], Steenken et al. [37]) and automated guided vehicles (AGVs, see Bae and Kim [1]). The case of multi-load AGVs (i.e.,

AGVs which can carry more than one container at a time) has been discussed by Grunow et al. [11]. Also the problem of allocating and scheduling stacking cranes has been considered (see Zhang et al. [44]). An integrated scheduling approach for automated stacking cranes and automated guided vehicles has been proposed by Meersmans and Wagelmans [31]. A method for sequencing the containers to be loaded onto a ship has been developed by Kim et al. [22]. Strategies for locating containers in the yard have been discussed for several problem settings (see de Castilho and Daganzo [5], Kim and Kim [23], Kim et al. [24], Taleb-Ibrahimi et al. [39], Zhang et al. [43]). In order to study the complex processes at container terminals with their dynamic nature, simulation models have been developed (see Gambardella et al. [8, 9], Kim et al. [25], Legato and Mazza [28], Yun and Choi [42]). A simulation study to compare different automated vehicle types at a container terminal has been provided by Vis and Harika [41]. Finally, a method to generate data for experiments with optimization and simulation approaches has been suggested (see Hartmann [15]). Detailed literature surveys have been given by Meersmans and Dekker [30] as well as Vis and Koster [40].

In this paper, we introduce a general model for scheduling container terminal resources such as different types of equipment and manpower. This way, we take an approach that is different from the scheduling papers listed above because we provide a model that is not designed for a single application. Our model consists of a set of jobs (which could be transportation tasks or other activities) that must be scheduled and assigned to a resource. For the temporal arrangement of the jobs, sequence-dependent setup times have to be observed. They can be used to cover, e.g., the empty times of the equipment between two container transports. The objective is to minimize the average lateness per job as well as the average setup time. In order to demonstrate the generality of our model, we present four applications from the port of Hamburg (Germany), namely straddle carriers, automated guided vehicles (AGVs), stacking cranes, and workers. Subsequently, we propose priority rule heuristics and a genetic algorithm for the general model and analyze them in a computational study. The paper closes with a summary of the results and discusses opportunities for future research.

# 2 A General Model for Container Terminal Scheduling Problems

In this section, we propose a general model for scheduling problems in container terminal logistics. After the definition of the model, we outline how several real-world scheduling problems can be captured by the general model.

## 2.1 Model Formulation

We consider a set $J = \{1, \ldots, n\}$ of jobs that have to be carried out using a set $R = \{1, \ldots, m\}$ of identical resources. Each job $j \in J$ has to be executed on one of the resources. Since the resources are identical, each job can be performed on any of the resources. Each resource $r \in R$ can be occupied with only one job at a time.

Each job $j \in J$ has a processing time $p_j > 0$. Before job $j$ can actually be processed, a setup time is required. Let $i_j \in J$ denote the job that is executed on resource $r \in R$ immediately before job $j$. A sequence-dependent setup time $s_{i_j,j} \geq 0$ has to be taken into account before processing job $j$. Once started, a job must not be interrupted, that is, both the setup and the processing phase of a job must be carried out as a non-preemptive whole. These requirements can be summarized as $f_{i_j} + s_{i_j,j} + p_j \leq f_j$ where $f_j$ is the finish time of job $j \in J$. For the setup times, the following triangle inequality must hold: We assume $s_{ik} \leq s_{ij} + s_{jk}$ for any three jobs $i$, $j$, and $k$.

Some of the jobs may be related by precedence constraints. The predecessors of job $j$ are given by the set $P_j$ (analogously, the successors of job $j$ are given by the set $S_j$). The precedence relation between a job $j$ and a predecessor $i \in P_j$ is specified by a time lag $\tau_{ij} \geq 0$. The latter implies

that job $j$ must finish at least $\tau_{ij}$ units later than job $i$, that is, $f_i + \tau_{ij} \le f_j$. $\tau_{ij} = 0$ represents a so-called end-to-end constraint because job $j$ must not finish before job $i$ finishes. $\tau_{ij} = p_j$ is a so-called end-to-start constraint since job $j$ must not start before the finish time of job $i$. Note that also other types of precedence constraints can be expressed with the general time lags employed here (further details can be found in Bartusch et al. [2]).

Each job $j \in J$ is related to two specific time instants. First, time $d_j$ is the due date of job $j$. That is, job $j$ should be completed at or before time $d_j$. Completing job $j$ later than time $d_j$ is allowed, but it will lead to penalty costs in the objective function. Second, time $e_j$ denotes the earliest time at which resource $r \in R$ that carries out job $j$ is available again after the execution of job $j$. Hence, if job $j$ is completed earlier than time $e_j$, then resource $r$ becomes available at time $e_j$. Otherwise, resource $r$ becomes available immediately after the completion of job $j$. Consider the finish time $f_j$ of a job $j$ and the earliest availability time $e_{i_j}$ of the job $i_j$ that is executed immediately before job $j$ on the same resource. We can summarize the restriction imposed by the earliest availability time by $e_{i_j} + s_{i_j,j} + p_j \le f_j$.

We assume that the model is applied repeatedly in a rolling planning horizon. Therefore, the scheduling problem must take the initial state of each resource into account, that is, the availability time and the setup state of that resource. The initial state of a resource is characterized by the last job carried out by that resource. For each resource $r \in R$, we denote the last job as $j_r^+$. This last job is a dummy job in the sense that it is assumed to be fixed, that is, it cannot be (re-)scheduled. The last jobs are comprised in the set $J^+ = \{j_r^+ | r \in R\}$, for which we assume $J \cap J^+ = \emptyset$. Each last job $j_r^+$ is associated with two types of information. First, it is related to a finish time $f_{j_r^+}$ that reflects the time at which resource $r$ is available. Note that $f_{j_r^+} \ge t_{\text{now}}$ must hold, where $t_{\text{now}}$ denotes the current time (without loss of generality, we assume $t_{\text{now}} = 0$). Second, job $j_r^+ \in J^+$ contains the initial setup state of resource $r \in R$ (hence the setup time $s_{ij}$ must be given for all $i \in J \cup J^+$ and $j \in J$). Let us emphasize again that only the jobs in $J$ (and not those in $J^+$) are considered for scheduling.

The problem now is to find a schedule for the jobs $j \in J$, that is, a resource $r_j$ and a finish time $f_j$ for each job $j \in J$ such that all constraints given above are observed. The objective is to minimize the weighted sum of average lateness per job and average setup time per job. The weight for the average lateness per job is denoted as $\alpha_L$ while $\alpha_S$ is the weight for the average setup time per job. Formally, the objective function can be given as

$$\text{minimize} \quad \alpha_L \cdot \frac{1}{n} \cdot \sum_{\substack{j \in J \\ f_j > d_j}} (f_j - d_j) \quad + \quad \alpha_S \cdot \frac{1}{n} \cdot \sum_{j \in J} s_{i_j,j} \ .$$

Note that lateness and setup times should be measured on the same scale (e.g., seconds). Lateness minimization and hence observing the due dates is one of the most important goals in practice. In the applications, setup times correspond to empty times of resources which should be minimized especially if the due dates are not tight.

The model contains several features that are well known from other scheduling problems in the literature. The concept of jobs with precedence constraints and resource requests can also be found in the classical resource-constrained project scheduling problem (see Pritsker et al. [35]). Time lags and different types of precedence constraints have been studied in connection with project scheduling as well (see Bartusch et al. [2]). Moreover, the resource assignment part of the new model is a special case of the multi-mode extension of the resource-constrained project scheduling problem (see Elmaghraby [7], Talbot [38]). The objective of minimizing the lateness with respect to due dates has also been considered in the context of project scheduling (see, e.g., Kapuscinska et al. [21]). The concept of sequence-dependent setup times occurs in many problems in the field of lotsizing and batching (see Jordan [20]). We will make use of the similarity between the new model and resource-constrained project scheduling when designing heuristics for the new model.

Of course, the model outlined above is a rather abstract formulation. The generality of the approach enables us to apply it to different problem settings. The jobs reflect tasks that have to be carried out in container handling. The resources can reflect the technical equipment for container handling or manpower. The setup and processing times can correspond to moving a resource to some location, moving a container, or other tasks. Several practical applications will be given in the following subsections.

In practice, all applications are embedded in an overall terminal control system. Once a resource (e.g., straddle carrier, automated guided vehicle, stacking crane, reefer worker) has completed a job, the related scheduling procedure should be executed, and the resource should be assigned its next job according to the new schedule. This means that rescheduling should be frequently done such that all decisions are based on the current data. Despite rescheduling before each job assignment, it should be useful to compute schedules with a horizon of several successive jobs. This is because a longer horizon increases the degrees of freedom on when to execute a job the due date of which is not tight. While short horizons would lead to a first-come-first-serve-like approach, longer horizons allow to exploit the possibility of starting a job at a time which leads to a shorter setup time.

Finally, note that the model structure with only a single resource type implies that we have a separate problem for each resource type. This allows to define a general model that is independent of the terminal configuration (integrated models with more than one resource type are of course possible and might be promising, but they would be much more dependent on the actual terminal and thus less general).

## 2.2 Application to Straddle Carriers

Many container terminals employ straddle carriers (sometimes also called van carriers) for transportation of containers on the terminal. Straddle carriers are used for transportation between the stack on the one hand and other locations such as a quai crane, the area for external trucks, or the train area on the other hand. Since straddle carriers are able to unload themselves and high enough to stack containers on top of each other, they can also be employed to transport containers to their positions within a yard block. Thus, they can also carry out shuffle moves within a block, that is, they can move a container to another position in case it stands on top of another container that must be moved out. If stacking cranes serve the blocks, however, straddle carriers transport containers from and to hand-over positions at a block, and they do not carry out shuffle moves.

Of course, the set of the transportation tasks of the straddle carriers defines the set of jobs for scheduling, and the number of resources is given by the number of straddle carriers. The processing time $p_j$ of a job $j$ is the time that a straddle carrier needs for the transportation of a container between two locations on the terminal. It is defined as the time between picking up the container and putting it down. As the locations are fixed, the transportation times are assumed as fixed as well. The setup time $s_{ij}$ between two jobs $i$ and $j$ is given by the time that a straddle carrier requires to get from the position where the container of job $i$ was put down to the pick-up position of job $j$. Hence, the setup time models the empty travel time of a straddle carrier. For any two positions, an estimate of the processing and setup time is usually available for scheduling. Of course, scheduling decisions can affect only the empty (or setup) times but not the transportation (or processing) times.

Precedence relations exist between jobs that correspond to containers that stand on top of each other in the stack. The job $i$ related to an upper container is a predecessor of the job $j$ related to the lower one (if the upper container is not demanded by a ship, truck, or train, job $i$ is a shuffle move). The time lag $\tau_{ij}$ is given by a buffer time which assures that the lower container can be picked up after the upper container has been moved away (although the latter job needs not necessarily be completed before the lower container is picked up).

The due date $d_j$ of a job $j$ is determined as follows. Let us first consider the case that a straddle carrier is supposed to bring a container from the stack to another location such as a quai crane

or an external truck. In this case, the due date reflects the latest time at which the container should have arrived there. For a quai crane waiting for a container, the due date corresponds to the time at which the quai crane requires the container in order to keep its loading sequence on time. Considering an external truck, the due date reflects the acceptable waiting time of the truck driver. Moreover, keeping the waiting times for trucks short helps to avoid congestions in the truck area. In the second case, a straddle carrier is supposed to bring a container to the stack. Here, the due date is used to model the latest time at which a straddle carrier has to pick up a container at some location. For example, an arriving container must be picked up from a quai crane or an external truck at or before the due date. In this case, we have a due date for pick-up, say $\bar{d}_j$. The latter can be transformed into a due date related to the completion of the job by setting $d_j = \bar{d}_j + p_j$. This is possible because the time between pick-up and completion is a constant, namely the transportation or processing time $p_j$. Recall that the due date is always related to the finish time of a job (see the objective function).

The earliest availability time $e_j$ of a resource after carrying out a job $j$ is used to model the blocking of straddle carriers by other resources. Consider a job $j$ that requires a straddle carrier to transport a container from a quai crane to a yard block. Let $\bar{e}_j$ denote the time at which the container is available for the straddle carrier (i.e., the time at which it has been put on the ground by the quai crane). The earliest release time of the straddle carrier is $e_j = \bar{e}_j + p_j$ which reflects the earliest possible time to complete this job. Now let us consider a job $j$ to transport a container from the stack to a truck which is already waiting for the container. In this case, we have $e_j = 0$ because the straddle carrier is available immediately after completing the job. These two cases should be sufficient to illustrate the use of the earliest release times. Generally speaking, we have $e_j > 0$ if the straddle carrier will have to wait for another resource, whereas we have $e_j = 0$ if it is not blocked by another resource.

The use of the initial state and of the objective function is obvious. The lateness component of the objective function attempts to serve the quai cranes (and also trucks) on time such that they do not have to wait. The setup time component leads to short empty travel times. Summing up, we have employed all features of the general scheduling model of Subsection 2.1 to capture the straddle carrier scheduling problem.

## 2.3   Application to Automated Guided Vehicles

On modern container terminals with a high degree of automation, often automated guided vehicles (AGVs) are employed to carry containers between the quai and the yard blocks. Unlike straddle carriers, AGVs are unable to unload themselves (automated vehicles with loading and unloading capability are often referred to as automated lifting vehicles, see Vis and Harika [41]—note that those vehicles can be modeled in the same way as straddle carriers). Therefore, stacking cranes are needed to serve the yard blocks. Hence, the jobs to schedule are the transportation of containers from a quai crane to a stacking crane and from a stacking crane to a quai crane. While the former case corresponds to the discharging of a vessel, the latter case occurs when loading a vessel.

The application of the general model to the AGV case is similar to the straddle carrier case with only a few differences. Again, the processing time $p_j$ reflects the transportation of a container while the setup time $s_{ij}$ models the empty time between two successive jobs. The due dates $d_j$ reflect the latest hand-over times at the quai cranes. If an AGV exceeds its due date, this will lead to a waiting time of the quai crane. Thus, keeping the due dates is crucial for a high quai crane productivity and hence for a short time in port for the vessels. The earliest release times $e_j$ consider the fact that AGVs can be blocked by the other terminal resources. Since AGVs cannot unload themselves, an AGV arriving before its due date will have to wait for the crane related to this job. Therefore, the earliest release time is equal to the due date, that is, $e_j = d_j$. Finally, precedence relations are needed to control the order of AGVs arriving at a quai crane with a container. This AGV order is important because the sequence of containers to be loaded onto a vessel is often fixed. The time lag $\tau_{ij}$ between two successive jobs $i$ and $j$ related to the same quai crane is given

by a small buffer time that allows the first AGV to leave the quai crane position (typically, one will have $0 < \tau_{ij} \le d_j - d_i$). Note that there are no precedence relations between jobs related to different quai cranes.

## 2.4 Application to Stacking Cranes

Usually, the stack is organized in several yard blocks. Many container terminals employ stacking cranes to serve these blocks. Normally, there is one crane for each block (occasionally, two cranes per block are used). So-called rail-mounted gantry cranes cannot be moved to another block. On the other hand, so-called rubber-tyred gantry cranes can be moved, but this takes a long time and is not done very often (decisions to transfer a crane are based on workload estimations of the blocks, see Zhang et al. [44]). Thus, for both crane types, crane assignments to blocks can be assumed to be fixed for the detailed scheduling problem considered here. Moreover, the assignment of containers to blocks is given (it is not a part of this scheduling problem). Therefore, we have a separate crane scheduling problem for each block. That is, we consider the set of jobs associated with a single block, and we have a single crane resource (or, in case of double cranes, two resources). The jobs include transportation moves between positions within the block on the one hand and AGVs, straddle carriers, or external trucks on the other hand. Furthermore, jobs representing shuffle moves have to be taken into account.

As for the straddle carrier case, we have precedence relations between jobs that correspond to containers that stand on top of each other. The processing time $p_j$ of a job $j$ is the transportation time that a crane needs between picking up the container and putting it down. Again, the setup time $s_{ij}$ between two jobs $i$ and $j$ is given by the time that the crane requires to get from the position where the container of job $i$ was put down to the pick-up position of job $j$. The due date $d_j$ of a job $j$ determines the latest acceptable completion time and hence also the waiting time of the other resource (AGV, straddle carrier) or the external truck. The earliest release times again model the blocking of the crane by other resources. For example, if a crane serves an AGV which arrives in a just-in-time fashion, the crane will not be available before the due date, that is, we have $e_j = d_j$. On the other hand, if a crane is to serve an external truck which has already arrived at the block, it will be available after completing the job, that is, we have $e_j = 0$.

## 2.5 Application to Reefer Workers

Reefer containers are used to carry goods that require a controlled temperature (such as refrigerated or frozen goods). They have a device for temperature regulation which requires electricity. Hence, they are stored in specific yard blocks or areas of blocks that provide electricity connections. Reefer containers require special handling by a manpower resource, the reefer workers.

The jobs carried out by the reefer workers are connecting arriving containers, disconnecting departing ones, doing small repair tasks, and controlling the temperature of the reefer containers in the stack. For each of these job types, a processing time $p_j$ is given. Note that, unlike the previously described applications, here the processing time does not correspond to moving to another location. The use of the setup times, however, is similar to equipment scheduling. The setup time $s_{ij}$ between a job $i$ and a job $j$ is the time that a reefer worker needs to move from the container associated with job $i$ to the container related to job $j$. An estimate of this time can be assumed to be available. This estimate is based on the container positions in the stack.

For connection jobs, the due date $d_j$ reflects the time that a reefer container is allowed to be without electricity. Of course, a container can keep its temperature in the allowed range only for a certain time if it is without electricity. For disconnection jobs, the due date reflects the latest time the container must be disconnected such that the stacking crane or straddle carrier can pick it up on time (the terminal control system might include a buffer time when computing the due date). For repair jobs and temperature control jobs, a due date is given as well. After a worker has completed a job, he can immediately move on to the next job. Therefore, we can set the earliest availability times to $e_j = 0$. A definition of precedence relations is not necessary.

In practice, the reefer workers are often equipped with portable radio data sets. When they have completed a job, they transmit this information to the terminal control system via the radio data set. In return, they get the next job.

# 3 Priority Rule Based Heuristics

## 3.1 Single Pass Dispatching Method

In this subsection, we present a simple dispatching heuristic for the container terminal scheduling problem. This method is a straightforward way to build job sequences for the resources. The following steps are repeated until all jobs have been scheduled:

- **Compute eligible jobs.** Compute the set $E$ of eligible jobs as the set of the currently unscheduled jobs of which all predecessors have already been scheduled.

- **Select job.** Select the job $j^*$ to be scheduled next as the eligible job with the smallest due date.

- **Select resource.** Select the resource $r^*$ that leads to the smallest increase in the objective function, that is, $r^*$ produces the smallest weighted sum of lateness and setup time for job $j^*$ among all resources.

- **Update schedule.** Schedule job $j^*$ at the end of the current job sequence of resource $r^*$.

The criteria for selecting a job and a resource can be seen as priority rules. According to the classification of Kolisch and Hartmann [27], this approach is a deterministic single-pass priority rule method. An alternative (and equally straightforward) rule for resource selection could be to select the resource with the earliest availability time in the current partial schedule. In Subsection 6.2, we carry out some computational experiments to compare both rules.

## 3.2 Multi Pass Sampling Method

The single pass priority rule based method described above produces only one schedule. We now attempt to improve the results by allowing for multiple passes. In each pass, a schedule is constructed. In order to produce different schedules, we randomize the minimum due date priority rule in the job selection process. We obtain a multi-pass biased random sampling method (see Kolisch and Hartmann [27]). Hence, the sampling heuristic is essentially a repeated application of the single pass method in which the job selection mechnism is modified.

The job selection mechanism is adapted as follows. We define a parameter $\delta$ which determines how many of the jobs in the eligible set $E$ are considered. Let $E_\delta$ denote the set of the $\delta$ eligible jobs with the smallest due dates in $E$. Now we select the eligible job $j^* \in E_\delta$ to be scheduled next on a biased random basis. The probability for eligible job $j \in E_\delta$ to be selected is given by

$$p(j) = \frac{d_{\max} - d_j + 1}{\sum_{i \in E_\delta}(d_{\max} - d_i + 1)},$$

where $d_{\max} = \max\{d_j \mid j \in E_\delta\}$ is the maximal due date. The resource selection mechanism is the same as in the single pass method. Once a schedule has been completed, the sampling heuristic proceeds to compute the next one. This is repeated until a time limit is reached, and the best schedule found is reported.

The priority rule based definition of the probabilities $p(j)$ implies that jobs with tighter due dates are more likely to be selected. This way, these jobs will be scheduled earlier, and the risk of exceeding their due dates is reduced. The parameter $\delta$ excludes jobs with non-tight due dates from consideration. Therefore, using small values for $\delta$ puts a focus on the jobs with the tightest due dates.

Two special cases of this sampling approach should be mentioned. $\delta \geq n$ induces conventional sampling in which the eligible set is not restricted. $\delta = 1$ implies that only the job with the smallest due date can be considered. Thus, in each pass, the same schedule is computed which is equal to the schedule found by the deterministic single pass approach.

# 4 Genetic Algorithm

Genetic algorithms (see Goldberg [10], Holland [17]) adopt the principles of biological evolution to solve hard optimization problems. For our genetic algorithm (GA), we exploit the similarities of the general container terminal scheduling problem to the resource-constrained project scheduling problem. These similarities allow us to use the project scheduling GA of Hartmann [13] as a starting point. This GA will be adapted to solve the problem introduced in this paper. In particular, the representation, the decoding procedure, and the construction of the initial population require problem-specific knowledge.

## 4.1 Basic Scheme

We apply the generational management framework of Eiben et al. [6]. The GA starts with the computation of an initial population, i.e., the first generation. The number of individuals in the population is referred to as $POP$. The GA then determines the fitness values of the individuals of the initial population. After that, we apply the crossover operator to produce new individuals ("children") from the existing ones ("parents"). Subsequently, we apply the mutation operator to the newly produced children. After computing the fitness of each child, we add the children to the current population. Then we apply the selection operator to reduce the population to its former size $POP$. Doing so, we obtain the next generation to which we again apply the crossover operator and so on. This process is repeated until a time limit is reached. Of course, other stopping criteria can be employed as well, e.g., a maximal number of generations or a number of generations without improvement of the best objective function value found so far. More formally, the GA scheme can be summarized as follows. Here, $\mathcal{P}$ denotes the current population (i.e., a set of individuals), and $\mathcal{C}$ is the set of children.

generate $POP$ individuals for the initial population $\mathcal{P}$;
apply decoding procedure to compute fitness for individuals $I \in \mathcal{P}$;
**while** time limit is not reached **do**
**begin**
    produce a set $\mathcal{C}$ of children from $\mathcal{P}$ by crossover;
    apply mutation to children $I \in \mathcal{C}$;
    apply decoding procedure to compute fitness for children $I \in \mathcal{C}$;
    $\mathcal{P} := \mathcal{P} \cup \mathcal{C}$;
    reduce population $\mathcal{P}$ to size $POP$ by means of selection;
**end.**

## 4.2 Problem Representation

Genetic algorithms for scheduling problems often do not operate directly on schedules but on representations of schedules. The latter are then transformed into schedules by means of a problem-specific decoding procedure. The advantage of such an indirect approach is that it allows to employ standard representations together with standard genetic operators (crossover, mutation).

In our genetic algorithm, a schedule is represented by a precedence feasible job list $(j_1, \ldots, j_n)$ and three additional genes $\beta_L$, $\beta_S$, and $\beta_W$, such that an individual $I$ is given as

$$I = (\, (j_1, \ldots, j_n)\, , \beta_L, \beta_S, \beta_W \,).$$

In a job list, each job appears exactly once, that is, we have $J = \{j_1, \ldots, j_n\}$. A job list is precedence feasible if all predecessors of a job $j$ appear in the list before job $j$, that is, $P_{j_i} \subseteq \{j_1, \ldots, j_{i-1}\}$ for $i = 1, \ldots, n$. A job list determines the order in which the jobs are scheduled by the decoding procedure. The job list representation is a generalization of the classical permutation based representation (see Reeves [36]). It has been shown to be superior to other representations for resource-constrained scheduling problems (see Hartmann [13, 14]).

While the representation controls the scheduling order, the task of assigning resources to jobs will be left to the decoding procedure. The remaining genes $\beta_L$, $\beta_S$, and $\beta_W$ are parameters that are used by the decoding procedure when selecting a resource for a job (see the following subsection). Note that using algorithm parameters in the genetic representation implies that they are exposed to evolution and survival-of-the-fittest (see Goldberg [10]). This is often referred to as self-adaptation.

## 4.3 Decoding Procedure and Fitness

The decoding procedure employs problem-specific features to transform the representation into a solution for the container terminal scheduling problem. It scans the job list from left to right and successively schedules the jobs $j_1, \ldots, j_n$ using the following steps:

- **Select job.** Select the next job $j_i$ from the job list.

- **Select resource.** Evaluate the partial schedules arising from assigning job $j_i$ to resource $r \in R$ as last job by computing the resulting lateness $l_{j_i}(r)$, setup time $s_{j_i}(r)$, and waiting time $w_{j_i}(r)$ of job $j_i$. A waiting time occurs if a resource completes a job earlier than the earliest release time, i.e., the resource would have to wait until the earliest release time before the next job can be started. These three times are weighted with the related genes of the individual, leading to $y_{j_i}(r) = \beta_L \cdot l_{j_i}(r) + \beta_S \cdot s_{j_i}(r) + \beta_W \cdot w_{j_i}(r)$. Now select a resource $r^*$ with the best evaluation $y_{j_i}(r^*) = \min\{y_{j_i}(r) \,|\, r \in R\}$.

- **Update schedule.** Schedule job $j_i$ at the end of the current job sequence of resource $r^*$.

While the scheduling order of jobs is prescribed by the job list representation, the decoding procedure takes care of the resource assignment part of the problem. The usage of the weight genes $\beta_L$, $\beta_S$, and $\beta_W$ for resource evaluation is based on two ideas. First, in the iterations of the decoding procedure, other settings than the overall objective function parameters might be useful. Second, they allow to penalize possible waiting times which might be worth avoiding during schedule computation because waiting times reduce the capacities that will be left for the jobs to be scheduled next. (Note, however, that considering waiting times in the overall objective function would not make much sense.) The fitness of an individual is defined as the objective function value of the schedule related to the individual.

## 4.4 Initial Population

Let us now define how to determine the first generation containing $POP$ individuals. For the construction of job lists, we employ the sampling strategy of Subsection 3.2. That is, in each step, we determine the restricted eligible set $E_\delta$ and draw a job $j \in E_\delta$ using the due date based probability $p(j)$. The selected job $j$ is then added at the end of the job list. Next, we draw the genes $\beta_L$, $\beta_S$, and $\beta_W$ using a parameter $\epsilon \geq 0$ which defines a range for the genes:

$$\beta_L \in [\max\{0, \alpha_L - \epsilon\}, \min\{\alpha_L + \epsilon, 1\}],$$

$$\beta_S \in [\max\{0, \alpha_S - \epsilon\}, \min\{\alpha_S + \epsilon, 1\}],$$

$$\beta_W \in [0, \epsilon].$$

Observe that the genes related to lateness and setup time are drawn from the $\epsilon$-neighborhood of the respective objective function weights (for the waiting time, there is no related weight in the objective function). Having determined these three genes, their values are normalized such that we obtain $\beta_L + \beta_S + \beta_W = 1$, that is, we set $\beta_X := \frac{\beta_X}{\beta_L + \beta_S + \beta_W}$ for $X \in \{L, S, W\}$.

Considering the job list construction, one might as well think of the following straightforward approach: The latter selects jobs on a pure random basis without due date bias, that is, with equal probabilities $p(j) = \frac{1}{|E|}$, where $E$ is the unrestricted eligible set. Using the sampling method instead should lead to more promising job lists while still maintaining reasonable genetic variety. The parameter $\epsilon$ controls the deviation of the weight genes from the objective function weights. Note that $\epsilon = 0$ reduces the resource evaluation to the respective objective function weights ($\beta_L = \alpha_L$ and $\beta_S = \alpha_S$) without considering waiting times ($\beta_W = 0$). The benefit of biased sampling and of the weight genes will be examined further in the computational tests of Subsection 6.3.

## 4.5 Crossover

For crossover, the current population is randomly partitioned into pairs of individuals. From each pair of individuals (parents), two new individuals (children) will be produced. Let us assume that two individuals of the current population have been selected for crossover. We have a mother individual $M = ((j_1^M, \ldots, j_n^M), \beta_L^M, \beta_S^M, \beta_W^M)$ and a father individual $F = ((j_1^F, \ldots, j_n^F), \beta_L^F, \beta_S^F, \beta_W^F)$. Now two child individuals have to be constructed, a daughter $D$ and a son $S$.

Let us start with a definition of the daughter $D = ((j_1^D, \ldots, j_n^D), \beta_L^D, \beta_S^D, \beta_W^D)$. Combining the parent's job lists, we have to make sure that each job appears exactly once in the daughter's job list. We make use of a general crossover technique presented by Reeves [36] for permutation based genotypes. For the one-point crossover, we draw a random integer $q$ with $1 \leq q \leq n$. The daughter's job list is determined by taking the job list of the positions $i = 1, \ldots, q$ from the mother, that is,

$$j_i^D := j_i^M .$$

The remaining positions $i = q + 1, \ldots, n$ are derived from the father. However, the jobs that have already been selected must not be considered again. The remaining jobs are taken in their relative order in the father's job list, that is, we set for $i = q + 1, \ldots, n$

$$j_i^D := j_k^F \quad \text{where} \quad k = \min\{1 \leq u \leq n \mid j_u^F \notin \{j_1^D, \ldots, j_{i-1}^D\}\}.$$

The three weight genes are taken from the mother, that is, we set

$$\beta_L^D := \beta_L^M , \;\; \beta_S^D := \beta_S^M , \;\; \beta_W^D := \beta_W^M.$$

The son individual is computed analogously. For the son's job list, the weight genes and the first part of the job list are taken from the father and the second part is taken from the mother.

In addition, we have tested a two-point crossover variant. Here, we draw two random integers $q_1$ and $q_2$ with $1 \leq q_1 < q_2 \leq n$. Analogously to the one-point variant, the weight genes and the first part of the job list until $q_1$ are taken from one parent. The second part of the job list between positions $q_1 + 1$ and $q_2$ is taken from the other parent, following the same logic as above. The third part of the list between $q_2 + 1$ and $n$ is again taken from the first parent.

This crossover strategy creates job lists in which each job appears exactly once. Moreover, it has been proven by Hartmann [13] that the resulting job lists are precedence feasible, given that the parents' job lists were precedence feasible as well. Since this crossover operator produces feasible offspring, there is no need for a repair operator. This property leads to a good inheritance behavior of building blocks of solutions. Also note that taking the weight genes and the first part of the job list from the same parent implies that the first part of the related schedule is inherited. This means that good partial schedules can be passed on to the offspring without being destroyed by crossover.

### 4.6   Mutation

The mutation operator is applied to each newly produced child individual. The probability for each gene to be mutated is denoted as $p_{\text{mutation}}$.

In the first step, we consider the mutation of the job list for which two alternatives have been considered. The swap variant is defined as follows. We move through the job list from left to right. Consider a current position $i \in \{1, \ldots, n-1\}$ in the job list

$$(j_1, \ldots, j_i, j_{i+1}, \ldots, j_n).$$

If $j_i$ is not a predecessor of $j_{i+1}$, we interchange these two successive jobs with probability $p_{\text{mutation}}$, which leads to a new job list

$$(j_1, \ldots, j_{i+1}, j_i, \ldots, j_n).$$

The shift variant can be described as follows. Again, we move through the job list from left to right. In this variant, however, we apply a right shift to each job with a probability of $p_{\text{mutation}}$. Consider a current position $i \in \{1, \ldots, n-1\}$ in the job list

$$(j_1, \ldots, j_i, \ldots, j_h, \ldots, j_z, \ldots, j_n).$$

Let $z$ be the smallest index of the successors of job $j_i$, that is, $z = \min\{k \mid j_k \in S_{j_i}\}$. Now job $j_i$ can be shifted behind some randomly drawn position $h \in \{i+1, \ldots, z-1\}$, which leads to job list

$$(j_1, \ldots, j_{i-1}, j_{i+1}, \ldots, j_h, j_i, j_{h+1}, \ldots, j_z, \ldots, j_n).$$

That is, job $j_i$ is right shifted within the job list and inserted immediately after some job $j_h$. Clearly, the resulting job list is still precedence feasible.

The second step considers the three parameters for the decoding procedure. Using again parameter $\epsilon$ (see Subsection 4.4), the operator randomly draws

$$\beta'_X \in [\max\{0, \beta_X - \epsilon\}, \min\{\beta_X + \epsilon, 1\}]$$

for $X \in \{L, S, W\}$. Again, each of these genes is mutated with probability $p_{\text{mutation}}$. Subsequently, the parameters are normalized as described in Subsection 4.4.

### 4.7   Selection

After the newly produced individuals have been added to the current population, the next step is to select the individuals that survive and make up the next generation. Following the study of Hartmann [13], we decided to employ the deterministic ranking method which follows a survival-of-the-fittest strategy (cf., e.g., Michalewicz [32]). This method sorts the individuals with respect to their fitness values and selects the $POP$ best ones while the remaining ones are deleted from the population (ties are broken arbitrarily).

## 5   Generating Experimental Data

### 5.1   Generator

In order to demonstrate the applicability of the general scheduling model and the genetic algorithm, we carried out several computational experiments. These experiments required test instances as input data for the heuristic. In order to obtain a large number of test instances, we developed a data generator for our problem setting. It is controlled by parameters that allow to produce instance sets with specific characteristics. In particular, the parameters enabled us to produce quite realistic test sets for the container terminal applications mentioned in Section 2.

The generator works as follows. The number $n$ of jobs and the number $m$ of resources are specified by the user as parameters. In addition to the $n$ regular jobs, the $m$ dummy jobs that

| Test set | number of jobs | number of resources | scheduling horizon |
|---|---|---|---|
| straddle carrier | 380 | 75 | 30 min |
| AGV | 100 | 50 | 15 min |
| reefer worker | 120 | 5 | 60 min |
| stacking crane | 8 | 1 | 30 min |

**Table 1:** Characteristics of the four generated test sets

represent the initial setup states for the resources are generated. Each (non-dummy) job $j$ is randomly assigned a processing time $p_j$ from $\{p_{\min}, \ldots, p_{\max}\}$, where $p_{\min}$ and $p_{\max}$ are parameters that denote the minimal and maximal processing time, respectively. Next, we generate a setup time $s_{ij}$ between two consecutive jobs $i$ and $j$ on the same resource. The following approach is designed to produce setup times which do not violate the triangle inequality. It employs two parameters $S_1$ and $S_2$, where $S_1$ can be interpreted as the minimal and $S_1 + S_2$ as the maximal setup time. First, each job $j$ is randomly assigned a value $y_j \in \{0, \ldots, S_2\}$. Then the setup time between jobs $i$ and $j$ is defined as $s_{ij} = S_1 + |y_j - y_i|$. On the basis of a parameter $T$ for the scheduling horizon, a due date $d_j$ is drawn from $\{S_1 + p_j, \ldots, T\}$ for each job $j$. The earliest release times are determined using a parameter $\rho \in [0, 1]$. This parameter allows to control whether the earliest release time $e_j$ of a job $j$ is equal to its due date or zero (recall that these were the two typical cases in the applications of our model). With probability $\rho$, we set $e_j = 0$ and with probability $1 - \rho$, we set $e_j = d_j$.

## 5.2   Test Sets

Using the generator described in the previous subsection, we generated a set of test instances for each of the four container terminal applications discussed in Section 2. That is, we have a straddle carrier set, an AGV set, a reefer worker set, and a stacking crane set. These four sets differ in the settings of the generator parameters and hence in their characteristics. We attempted to generate realistic instances which may similarly occur in practice at peak time on a medium-sized container terminal. The characteristics of the test sets can be summarized as follows (see also the main parameter settings displayed in Table 1):

- The straddle carrier set contains the largest instances. This is because we assume the straddle carriers to carry out transportation jobs both on the seaside and on the landside as well as shuffle moves. This set includes the largest number of resources and jobs and a medium scheduling horizon.

- The AGV case considers only transportation jobs between quai and stack. Therefore, we have selected smaller numbers of jobs and resources than for the straddle carrier set. Moreover, we have a shorter scheduling horizon because shuffle moves are not considered.

- In the reefer worker set, we have fewer resources than in the previous two cases but a longer scheduling horizon. The latter results from substantially longer time windows for reefer jobs.

- The stacking crane case is the smallest problem setting with only one resource (corresponding to one yard block served by a single crane). Due to shuffle moves and landside operations with less tight due dates, the horizon is the same as for the straddle carrier set.

For each of the test sets, the parameter settings were done in a way that leads to scarce resources. Scarce resources make it difficult to find schedules with small lateness—thus, we obtain challenging test problems. In fact, the parameters were adjusted such that we have an average lateness $> 0$ even for the best heuristic. This allows for a meaningful comparison of heuristics on the basis of the objective function values. Moreover, the case of scarce resources is particularly

| Problem | heuristic | objective | lateness | setup time | late jobs |
|---|---|---|---|---|---|
| Straddle carrier | dispatching | 31.0 | 27.3 s | 63.9 s | 27.5% |
| | sampling | 26.2 | 22.4 s | 61.9 s | 24.8% |
| | GA | 10.4 | 4.6 s | 62.1 s | 15.9% |
| AGV | dispatching | 22.8 | 17.6 s | 69.2 s | 20.5% |
| | sampling | 18.4 | 13.0 s | 67.3 s | 18.6% |
| | GA | 13.7 | 7.4 s | 69.8 s | 19.6% |
| Reefer worker | dispatching | 87.4 | 89.4 s | 69.1 s | 46.3% |
| | sampling | 13.1 | 9.5 s | 45.5 s | 10.3% |
| | GA | 8.8 | 5.8 s | 34.9 s | 7.4% |
| Stacking crane | dispatching | 30.1 | 28.1 s | 47.6 s | 30.1% |
| | sampling | 12.4 | 9.4 s | 38.8 s | 13.0% |
| | GA | 12.4 | 9.5 s | 38.4 s | 13.1% |

**Table 2:** Comparison of the heuristics (time limit: 1 s on Pentium 4 with 1.6 Ghz)

important in practice. In this case, minimization of lateness is crucial for successfully carrying out the logistic processes. For each of the four cases, 250 instances were generated. Hence, the test set consists of 1,000 instances altogether which should be a reasonable basis for a thorough computational analysis.

## 6  Computational Results

### 6.1  Comparison of the Heuristics

In order to evaluate the heuristics, they were coded in ANSI C and compiled with the lcc compiler. The experiments were carried out on a Pentium 4-m based computer with 1.6 GHz running under Windows XP. In order to consider the real-time application of the scheduling model which requires very short computation times, a time limit of one second has been selected. Recall that rescheduling should be done after a resource has completed its last job and before it is assigned its next job, such that the computation time is waiting time for the resource.

The weights of the objective function have been set to $\alpha_L = 0.9$ and $\alpha_S = 0.1$ which should be a reasonable choice in practice. Meeting a due date is considered to be more important than setup time minimization (also, minimizing lateness will probably lead to small setup times). In case of small workload and thus plenty resource capacities, the lateness criterion becomes less critical and the setup time minimization plays a more important role.

Table 2 gives the results for the single pass dispatching method of Subsection 3.1, the sampling heuristic of Subsection 3.2, and the genetic algorithm of Section 4. The results are given separately for each of the four problem sets of Table 1. For each heuristic, we report the average objective function value together with the average lateness per job and the average setup time per job (both in seconds). The average percentage of late jobs is also displayed.

For the straddle carrier, AGV, and reefer worker cases, the sampling method performs better than the single pass approach while the GA outperforms both. In the stacking crane case, the sampling method and the GA produce similar results which is due to the small instance size. Also recall that we have an average lateness $> 0$ due to the design of the test instances. Considering the straddle carrier and the AGV cases, the GA reduces the lateness while all methods lead to similar setup times. In the reefer worker case, the GA reduces both the lateness and the setup times. Note that the long scheduling horizon in the reefer case leads to a greater optimization potential than in the other cases. The GA appears to be best suited for exploiting this optimization potential.

| Problem | priority rule | objective | lateness | setup time | late jobs |
|---|---|---|---|---|---|
| Straddle carrier | objective increase | 31.0 | 27.3 s | 63.9 s | 27.5% |
| | earliest available resource | 360.5 | 375.4 s | 226.4 s | 86.3% |
| AGV | objective increase | 22.8 | 17.6 s | 69.2 s | 20.5% |
| | earliest available resource | 153.1 | 145.1 s | 224.9 s | 68.9% |
| Reefer worker | objective increase | 87.4 | 89.4 s | 69.1 s | 46.3% |
| | earliest available resource | 1367.5 | 1497.0 s | 201.6 s | 98.3% |

**Table 3:** Impact of priority rules for resource selection (deterministic single pass dispatching)

Assuming that the test instances are quite realistic, we can state that these results suggest to apply the GA in practice.

Furthermore, recall that the GA uses priority rules to compute initial solutions and then proceeds with evolutionary inheritance. This leads to better results than employing these priority rules over the entire computation time, even if the computation time is rather short. This observation is in line with studies on other scheduling problems (see [16]), but it should be noted that such a superior performance of the evolutionary inheritance mechanism is only possible with an appropriate genetic representation (see also [13]).

The following subsections deal with the configuration of the heuristics and with the effect of the algorithm parameters. The stacking crane test set will not be considered because the instances are too small for a meaningful analysis.

## 6.2    Configuration of the Priority Rule Methods

We start the analysis with a comparison of the priority rules for resource selection in the deterministic single pass heuristic of Subsection 3.1. The results given in Table 3 show that the rule based on the increase in the objective function value leads to much smaller lateness and setup times than the alternative rule. This demonstrates that a seemingly reasonable rule as the earliest availability time criterion may in fact produce clearly inferior results. Due to this result, the objective increase rule is the standard rule in the following experiments with the sampling method (note that it has also been used to produce the results of Table 2).

Next, we examine the sampling approach in more detail. Again, a time limit of one second is used. We analyze the impact of parameter $\delta$ which determines the number of eligible jobs considered for sampling (cf. Subsection 3.2). Table 4 displays the sampling results for different settings of $\delta$. A value of $\delta = 10$ appears to be a good overall choice. Thus, it was selected as the standard setting which was used to produce the sampling results of Table 2. Also observe that this sampling approach is better than conventional sampling ($\delta = n$) and deterministic single pass scheduling ($\delta = 1$). Hence, the restriction of the eligible set for sampling is a good strategy for this problem setting.

## 6.3    Configuration of the Genetic Algorithm

In this subsection, we study the impact of the genetic algorithm parameters and provide the best configuration. Table 5 gives the computational results for various parameter settings. It is divided into three blocks corresponding to the three problem sets of straddle carriers, AGVs, and reefer workers. We examine the population size $POP$, the method to construct an initial population (simple random or priority based sampling), the number of crossover points $C$ (1 or 2), the mutation rate $p_{\mathrm{mut}}$ (between 0.01 and 0.1), the mutation operator (swap or shift), and the range $\epsilon$ for the decoding procedure genes (between 0 and 1). The first row for each problem case provides the best parameter settings (this is the standard setting that was used to produce the

| Problem | $\delta$ | objective | lateness | setup time | late jobs |
|---|---|---|---|---|---|
| Straddle carrier | $n$ | 133.7 | 141.5 s | 63.6 s | 43.5% |
| | 20 | 26.2 | 22.4 s | 62.0 s | 24.3% |
| | 10 | 26.2 | 22.4 s | 61.9 s | 24.8% |
| | 5 | 27.0 | 23.1 s | 62.2 s | 25.2% |
| | 1 | 31.0 | 27.3 s | 63.9 s | 27.5% |
| AGV | $n$ | 28.3 | 23.9 s | 68.1 s | 23.4% |
| | 20 | 18.2 | 12.8 s | 67.1 s | 18.4% |
| | 10 | 18.4 | 13.0 s | 67.3 s | 18.6% |
| | 5 | 19.4 | 14.0 s | 67.5 s | 19.0% |
| | 1 | 22.8 | 17.6 s | 69.2 s | 20.5% |
| Reefer worker | $n$ | 132.9 | 141.9 s | 52.2 s | 29.2% |
| | 20 | 21.0 | 18.1 s | 47.4 s | 12.9% |
| | 10 | 13.1 | 9.5 s | 45.5 s | 10.3% |
| | 5 | 13.3 | 9.7 s | 45.4 s | 11.7% |
| | 1 | 87.4 | 89.4 s | 69.1 s | 46.3% |

**Table 4:** Configuration of the sampling heuristic (time limit: 1 s on Pentium 4 with 1.6 Ghz)

of Table 2). The following rows of each block contain parameter variations which are underlined. Note that the values for the population size $POP$ have been chosen with respect to the actual problem size (the more jobs and resources we have, the longer is the computation time for one schedule, which makes smaller populations within the same time limit reasonable). The standard settings for the other parameters are the same for all problem sets.

As shown in Table 5, some parameters have a significant impact on the GA results. First, the initial population should be produced by the sampling approach instead of a straightforward random assignment. The impact of sampling is stronger for longer scheduling horizons (straddle carrier and reefer worker cases). For long horizons, random job lists are likely to have jobs with early due dates in a position at the end of the list, which means that they are considerably late when they are scheduled. Second, the mutation operator should employ swaps rather than shifts. Shifting a job by many positions bears the risk that it will be scheduled much too late to meet its due date. Such a negative effect of the shift operator occurs particularly for the long horizon associated with the reefer worker case. Third, a range of $\epsilon > 0$ for the decoding procedure genes has a positive effect (recall that this means that resources are selected using adapted weights instead of the original weights of the objective function). This holds particularly for the straddle carrier and AGV cases where it pays to avoid waiting times which occur if a resource completes a job before its earliest release time. Note that in the reefer worker case waiting times cannot occur because there are no earliest release times to be considered. The remaining parameters have only a very small impact. This means that the GA is robust in the sense that its performance does not deteriorate if those parameters are chosen according to a rule of thumb.

## 6.4 Further Results

For the sampling method and the GA, Table 6 reports the average number of schedules that is computed within the time limit of one second (on a Pentium 4 with 1.6 Ghz). For the GA, also the number of generations and the population size are given. Generally speaking, the GA appears to produce a sufficient number of schedules within the short time limit to allow for a successful evolution. This holds in particular since GAs with good initial solutions require less iterations to produce near-optimal solutions than GAs which start from random solutions (recall that our GA employs the priority rule based sampling method to compute the initial generation).

| Problem | $POP$ | initial | $C$ | $p_{\mathrm{mut}}$ | mutation | $\epsilon$ | objective | lateness | setup | late jobs |
|---|---|---|---|---|---|---|---|---|---|---|
| Straddle | 40 | sampling | 2 | 0.05 | swap | 0.5 | 10.4 | 4.6 s | 62.1 s | 15.9% |
| carrier | <u>20</u> | sampling | 2 | 0.05 | swap | 0.5 | 10.4 | 4.6 s | 62.6 s | 16.0% |
|  | <u>60</u> | sampling | 2 | 0.05 | swap | 0.5 | 10.4 | 4.6 s | 62.4 s | 16.1% |
|  | 40 | <u>random</u> | 2 | 0.05 | swap | 0.5 | 104.9 | 109.1 s | 67.0 s | 37.6% |
|  | 40 | sampling | <u>1</u> | 0.05 | swap | 0.5 | 10.5 | 4.7 s | 62.3 s | 16.0% |
|  | 40 | sampling | 2 | <u>0.01</u> | swap | 0.5 | 10.4 | 4.6 s | 62.2 s | 16.1% |
|  | 40 | sampling | 2 | <u>0.10</u> | swap | 0.5 | 10.4 | 4.6 s | 62.3 s | 16.0% |
|  | 40 | sampling | 2 | 0.05 | <u>shift</u> | 0.5 | 11.0 | 5.3 s | 62.7 s | 16.7% |
|  | 40 | sampling | 2 | 0.05 | swap | <u>0.0</u> | 25.2 | 21.2 s | 61.5 s | 24.0% |
|  | 40 | sampling | 2 | 0.05 | swap | <u>1.0</u> | 10.6 | 4.8 s | 62.4 s | 16.9% |
| AGV | 60 | sampling | 2 | 0.05 | swap | 0.5 | 13.7 | 7.4 s | 69.8 s | 19.6% |
|  | <u>40</u> | sampling | 2 | 0.05 | swap | 0.5 | 13.7 | 7.5 s | 69.9 s | 19.7% |
|  | <u>80</u> | sampling | 2 | 0.05 | swap | 0.5 | 13.7 | 7.6 s | 69.7 s | 19.7% |
|  | 60 | <u>random</u> | 2 | 0.05 | swap | 0.5 | 19.3 | 13.1 s | 75.5 s | 26.3% |
|  | 60 | sampling | <u>1</u> | 0.05 | swap | 0.5 | 13.7 | 7.4 s | 69.8 s | 19.7% |
|  | 60 | sampling | 2 | <u>0.01</u> | swap | 0.5 | 13.8 | 7.5 s | 70.0 s | 19.7% |
|  | 60 | sampling | 2 | <u>0.10</u> | swap | 0.5 | 13.7 | 7.4 s | 70.0 s | 19.7% |
|  | 60 | sampling | 2 | 0.05 | <u>shift</u> | 0.5 | 13.7 | 7.4 s | 69.9 s | 19.8% |
|  | 60 | sampling | 2 | 0.05 | swap | <u>0.0</u> | 16.8 | 11.3 s | 66.1 s | 19.6% |
|  | 60 | sampling | 2 | 0.05 | swap | <u>1.0</u> | 14.1 | 7.8 s | 69.8 s | 19.7% |
| Reefer | 120 | sampling | 2 | 0.05 | swap | 0.5 | 8.8 | 5.8 s | 34.9 s | 7.4% |
| worker | <u>80</u> | sampling | 2 | 0.05 | swap | 0.5 | 8.8 | 6.0 s | 34.6 s | 7.6% |
|  | <u>160</u> | sampling | 2 | 0.05 | swap | 0.5 | 8.8 | 5.9 s | 35.4 s | 7.6% |
|  | 120 | <u>random</u> | 2 | 0.05 | swap | 0.5 | 137.5 | 148.5 s | 38.4 s | 28.6% |
|  | 120 | sampling | <u>1</u> | 0.05 | swap | 0.5 | 9.2 | 6.3 s | 35.6 s | 7.7% |
|  | 120 | sampling | 2 | <u>0.01</u> | swap | 0.5 | 9.1 | 6.2 s | 34.9 s | 7.6% |
|  | 120 | sampling | 2 | <u>0.10</u> | swap | 0.5 | 8.8 | 5.8 s | 35.2 s | 7.4% |
|  | 120 | sampling | 2 | 0.05 | <u>shift</u> | 0.5 | 11.4 | 8.0 s | 42.1 s | 9.1% |
|  | 120 | sampling | 2 | 0.05 | swap | <u>0.0</u> | 8.9 | 6.1 s | 34.6 s | 7.4% |
|  | 120 | sampling | 2 | 0.05 | swap | <u>1.0</u> | 8.9 | 6.0 s | 35.2 s | 7.5% |

**Table 5:** Configuration of the genetic algorithm (time limit: 1 s on Pentium 4 with 1.6 Ghz)

Furthermore, in both heuristics, the computation time for one schedule increases with the number of jobs and the number of resources. Hence, the number of schedules computed within the time limit depends of the problem set. It is also interesting to note that the number of computed schedules is different for the two approaches. The GA includes additional effort for crossover and selection. On the other hand, the job selection is faster in the GA where the next job is simply picked from the chromosome. In the sampling method, the eligible jobs have to computed, and a randomized selection mechanism is applied. Therefore, if the number of resources is small (as in the reefer worker case), the job selection makes up for a large part of the computational effort in the sampling heuristic. Thus, the GA produces more schedules within the same time limit.

# 7  Conclusions and Research Perspectives

In this paper, we proposed a general optimization model for scheduling jobs at container terminals. We showed that our model is applicable to straddle carriers, automated guided vehicles, stacking cranes, and reefer workers. The generality of our model is advantageous in practice because it allows to use the same model and optimization algorithms for several different scheduling problems. Furthermore, we developed priority rule heuristics and a genetic algorithm to solve the proposed problem. With a tailored instance generator, we generated several large sets of test instances for a

| Problem | sampling | GA | | |
|---|---|---|---|---|
| | #schedules | #schedules | #generations | population size |
| Straddle carrier | 321 | 320 | 8 | 40 |
| AGV | 1903 | 2047 | 35 | 60 |
| Reefer worker | 5172 | 7524 | 63 | 120 |

**Table 6:** Average number of schedules computed within the time limit (1 s)

computational analysis. The experiments showed that the genetic algorithm leads to better results than the priority rule methods. It appears to be well suited to solve test instances of realistic size within very short computation times. This makes it applicable to online scheduling within a terminal control system.

An important topic for future research is the integration of the optimization approach proposed in this paper into a simulation model. Simulation models cover the behavior of the equipment types as well as the control and optimization strategies. Providing a dynamic environment, they allow to capture the logistic processes in a realistic way. Simulation models have various applications ranging from the assessment of equipment capacities and layout alternatives to tests of optimization components. In particular, they provide a more realistic test bed for scheduling approaches than an offline study like the one that was carried out in this paper. On container terminals, the typical overall objectives are to maximize the productivity (i.e., the number of containers handled per hour) and to minimize the ships' times in port. Such objectives are not applicable to equipment scheduling. Therefore, an objective like the minimization of lateness and setup times is often chosen for scheduling. In order to analyze whether the scheduling objective really leads to good overall productivity and short times in port, a simulation model is needed. In addition, simulation models allow to examine various other points such as the impact of inaccurate estimates of processing times or delays (and resulting updates of due dates or availability times) as well as the coordination of different resource types. Thus, the scheduling approach proposed in this paper can only be fully evaluated by integrating it into a simulation model. While the experiments presented here showed that the GA produces good results in terms of the scheduling objective, the next step would be to examine whether the scheduling objective leads to a good terminal productivity. Note, however, that a simulation study always depends on the container terminal under consideration since it captures the actual equipment and the design of the terminal control system. Therefore, a simulation model cannot be used to evaluate a scheduling approach in a way that leads to general results (i.e., results which hold for any terminal configuration). The scheduling model suggested in this paper has been successfully tested in a simulation study for a real-world container terminal (for reasons of confidence, however, details on that study cannot be given).

# References

[1] J. W. Bae and K. H. Kim. A pooled dispatching strategy for automated guided vehicles in port container terminals. *International Journal of Management Science*, 6:47–70, 2000.

[2] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.

[3] J. Böse, T. Reiners, D. Steenken, and S. Voß. Vehicle dispatching at seaport container terminals using evolutionary algorithms. In R. H. Sprague, editor, *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 377–388. IEEE, Piscataway, 2000.

[4] C. F. Daganzo. The crane scheduling problem. *Transportation Research B*, 23:159–175, 1989.

[5] B. de Castilho and C. F. Daganzo. Handling strategies for import containers at marine terminals. *Transportation Research B*, 27:151–166, 1993.

[6] A. E. Eiben, E. H. L. Aarts, and K. M. van Hee. Global convergence of genetic algorithms: A markov chain analysis. *Lecture Notes in Computer Science*, 496:4–12, 1990.

[7] S. E. Elmaghraby. *Activity networks: Project plan-*

*ning and control by network models.* Wiley, New York, 1977.

[8] L. M. Gambardella, G. Bontempi, E. Taillard, D. Romanengo, G. Raso, and P. Piermari. Simulation and forecasting of an intermodal container terminal. In A. G. Bruzzone and E. J. H. Kerckhoffs, editors, *Simulation in Industry – Proceedings of the 8th European Simulation Symposium*, pages 626–630. SCS, Ghent, Belgium, 1996.

[9] L. M. Gambardella, A. E. Rizzoli, and M. Zaffalon. Simulation and planning of an intermodal container terminal. *Simulation*, 21:107–116, 1998.

[10] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[11] M. Grunow, H.-O. Günther, and M. Lehmann. Dispatching multi-load AGVs in highly automated seaport container terminals. *OR Spectrum*, 26:211–235, 2004.

[12] Y. Guan and R. K. Cheung. The berth allocation problem: Models and solution methods. *OR Spectrum*, 26:75–92, 2004.

[13] S. Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45:733–750, 1998.

[14] S. Hartmann. Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, 102:111–135, 2001.

[15] S. Hartmann. Generating scenarios for simulation and optimization of container terminal logistics. *OR Spectrum*, 26:171–192, 2004.

[16] S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127:394–407, 2000.

[17] H. J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[18] A. Imai, K. Nagaiwa, and C. W. Tat. Efficient planning of berth allocation for container terminals in Asia. *Journal of Advanced Transportation*, 31:75–94, 1997.

[19] A. Imai, E. Nishimura, and S. Papadimitriou. The dynamic berth allocation problem for a container port. *Transportation Research B*, 35:401–417, 2001.

[20] C. Jordan. *Batching and scheduling – Models and methods for several problem classes.* Number 437 in Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, Germany, 1996.

[21] T. T. Kapuscinska, T. E. Morton, and P. Ramnath. High-intensity heuristics to minimize weighted tardiness in resource-constrained multiple dependent project scheduling. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1998.

[22] K. H. Kim, J. S. Kang, and K. R. Ryu. A beam search algorithm for the load sequencing of outbound containers in port container terminals. *OR Spectrum*, 26:93–116, 2004.

[23] K. H. Kim and H. B. Kim. Segregating space allocation models for container inventories in port container terminals. *International Journal of Production Economics*, 59:415–423, 1999.

[24] 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*, 124:89–101, 2000.

[25] K. H. Kim, S. H. Won, J. K. Lim, and T. Takahashi. A simulation-based test-bed for a control software in automated container terminals. In *Proceedings of the International Conference on Computers and Industrial Engineering*, pages 239–243. Montreal, Canada, 2001.

[26] K. Y. Kim and K. H. Kim. A routing algorithm for a single straddle carrier to load export containers onto a container ship. *International Journal of Production Economics*, 59:425–433, 1999.

[27] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Węglarz, editor, *Project scheduling: Recent models, algorithms and applications*, pages 147–178. Kluwer Academic Publishers, 1999.

[28] P. Legato and R. M. Mazza. Berth planning and resources optimisation at a container terminal via discrete event simulation. *European Journal of Operational Research*, 133:537–547, 2001.

[29] A. Lim. The berth planning problem. *Operations Research Letters*, 22:105–110, 1998.

[30] P. J. M. Meersmans and R. Dekker. Operations research supports container handling. Technical Report EI 2001-22, Econometric Institute, Erasmus University Rotterdam, 2001.

[31] P. J. M. Meersmans and A. P. M. Wagelmans. Effective algorithms for integrated scheduling of handling equipment at automated container terminals. Technical Report EI 2001-19, Econometric Institute, Erasmus University Rotterdam, 2001.

[32] Z. Michalewicz. Heuristic methods for evolutionary computation techniques. *Journal of Heuristics*, 1:177–206, 1995.

[33] Y.-M. Park and K. H. Kim. A scheduling method for berth and quai cranes. *OR Spectrum*, 25:1–23, 2003.

[34] R. I. Peterkofsky and C. F. Daganzo. A branch and bound solution method for the crane scheduling problem. *Transportation Research B*, 24:159–172, 1990.

[35] A. A. B. Pritsker, L. J. Watters, and P. M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–107, 1969.

[36] C. R. Reeves. Genetic algorithms and combinatorial optimization. In V. J. Rayward-Smith, editor, *Applications of modern heuristic methods*, pages 111–125. Alfred Waller Ltd., Henley-on-Thames, 1995.

[37] D. Steenken, A. Henning, S. Freigang, and S. Voß. Routing of straddle carriers at a container terminal with the special aspect of internal moves. *OR Spectrum*, 15:167–172, 1993.

[38] F. B. Talbot. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28:1197–1210, 1982.

[39] M. Taleb-Ibrahimi, B. de Castilho, and C. F. Daganzo. Storage space vs. handling work in container terminals. *Transportation Research B*, 27:13–32, 1993.

[40] I. F. A. Vis and R. de Koster. Transshipment of containers at a container terminal: An overview. *European Journal of Operational Research*, 147:1–16, 2003.

[41] I. F. A. Vis and I. Harika. A comparison of vehicle types for automated container terminals. *OR Spectrum*, 26:117–143, 2004.

[42] W. Y. Yun and Y. S. Choi. A simulation model for container-terminal operation analysis using an object-oriented approach. *International Journal of Production Economics*, 59:221–230, 1999.

[43] C. Zhang, J. Liu, Y.-W. Wan, K. G. Murty, and R. J. Linn. Storage space allocation in container terminals. *Transportation Research B*, 37:883–903, 2003.

[44] C. Zhang, Y.-W. Wan, J. Liu, and R. J. Linn. Dynamic crane deployment in container storage yards. *Transportation Research B*, 36:537–555, 2002.