DYNAMIC SCHEDULING OF HANDLING EQUIPMENT AT AUTOMATED CONTAINER TERMINALS

PATRICK J.M. MEERSMANS AND ALBERT P.M. WAGELMANS

ERIM REPORT SERIES *RESEARCH IN MANAGEMENT*	
ERIM Report Series reference number	ERS-2001-69-LIS
Publication	November 2001
Number of pages	20
Email address corresponding author	meersmans@few.eur.nl
Address	Erasmus Research Institute of Management (ERIM) Rotterdam School of Management / Faculteit Bedrijfskunde Erasmus Universiteit Rotterdam P.O. Box 1738 3000 DR Rotterdam, The Netherlands Phone: +31 10 408 1182 Fax: +31 10 408 9640 Email: info@erim.eur.nl Internet: www.erim.eur.nl

BIBLIOGRAPHIC DATA AND CLASSIFICATIONS

Abstract	In this paper we consider the problem of integrated scheduling of various types of handling equipment at an automated container terminal in a dynamic environment. This means that the handling times are not known exactly beforehand and that the order in which the different pieces of equipment handle the containers need not be specified completely in advance. Instead, (partial) schedules may be updated when new information on realizations of handling times becomes available. We present an optimization based Beam Search heuristic and several dispatching rules. An extensive computational study is carried out to investigate the performance of these solution methods under different scenarios. The main conclusion is that, in our tests, the Beam Search heuristic performs best on average, but that some of the relatively simple dispatching rules perform almost as good. Furthermore, our study indicates that it is effective important to base a planning on a long horizon with inaccurate data, than to update the planning often in order to take newly available information into account.	
Library of Congress Classification (LCC)	5001-6182	Business
	5201-5982	Business Science
	HD 69.T54	Scheduling
Journal of Economic Literature (JEL)	M	Business Administration and Business Economics
	M 11 R 4	Production Management Transportation Systems
	C 69	Mathematical methods and programming: other
European Business Schools Library Group (EBSLG)	85 A	Business General
	260 K 240 B	Logistics Information Systems Management
	250 260	Mathematics Transportation
Gemeenschappelijke Onderwerpsontsluiting (GOO)		
Classification GOO	85.00	Bedrijfskunde, Organisatiekunde: algemeen
	85.34 85.20	Logistiek management Bestuurlijke informatie, informatieverzorging
	85.03	Methoden en Technieken, operations research
Keywords GOO	Bedrijfskunde / Bedrijfseconomie	
	Bedrijfsprocessen, logistiek, management informatiesystemen	
	Scheduling, Containervervoer, Algoritmen	
Free keywords	Container terminal, dynamic scheduling, beam search, dispatching rules	

Dynamic scheduling of handling equipment at automated container terminals

Patrick J.M. Meersmans[*] Albert P.M. Wagelmans[†]

November 21, 2001

Abstract

In this paper we consider the problem of integrated scheduling of various types of handling equipment at an automated container terminal in a dynamic environment. This means that the handling times are not known exactly beforehand and that the order in which the different pieces of equipment handle the containers need not be specified completely in advance. Instead, (partial) schedules may be updated when new information on realizations of handling times becomes available. We present an optimization based Beam Search heuristic and several dispatching rules. An extensive computational study is carried out to investigate the performance of these solution methods under different scenarios. The main conclusion is that, in our tests, the Beam Search heuristic performs best on average, but that some of the relatively simple dispatching rules perform almost as good. Furthermore, our study indicates that it is effective important to base a planning on a long horizon with inaccurate data, than to update the planning often in order to take newly available information into account.
Keywords: container terminal, dynamic scheduling, Beam Search, dispatching rules

1 Introduction

The handling of containers at seaport terminals becomes more and more automated. In 1993, the first automated container terminal in world was put into operation in Rotterdam, the Netherlands. This terminal uses both Automated Stacking Cranes (ASC) for the retrieval of containers from the stack and Automated Guided Vehicles (AGV) for the transport of containers to the Quay Cranes (QC) that load the containers into the vessel. Since the operating costs of the vessels handled at this terminal are very high (up to \$1000 an hour), the loading and unloading has to be done rapidly. Efficient scheduling of the automated handling equipment is crucial to achieve this.

AGV systems also exists within the area of Flexible Manufacturing Systems (FMS). In this context, several authors have considered the scheduling of AGV's. We mention Egbelu & Tanchoco (1984) who were among the first to investigate the performance of dispatching rules. However,

[*]Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, NL–3000 DR Rotterdam, The Netherlands; e-mail: `meersmans@few.eur.nl`.

[†]Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, NL–3000 DR Rotterdam, The Netherlands; e-mail: `wagelmans@few.eur.nl`.

such decision rules cannot be applied in a straightforward way to the scheduling of AGV's at container terminals. This is due to *blocking,* i.e., the fact that the AGV's always require a crane (ASC or QC) for loading and unloading. Hence, uncoordinated scheduling of the AGV's may result in low performance or even deadlock situations. Therefore, in Meersmans & Wagelmans (2001) we proposed an integrated approach to the scheduling of all terminal equipment (ASC's, AGV's and QC's). Both an exact Branch & Bound algorithm and a Beam Search heuristic were developed to solve the integrated scheduling problem in a static environment, i.e., it was assumed that all data is deterministic and known in advance. In this paper, we consider the dynamic case in which the handling times are not known exactly beforehand and the order in which the different pieces of equipment handle the containers need not be specified in advance. Instead, (partial) schedules may be updated when new information about realizations of handling times becomes available. We consider a planning method that uses the Beam Search heuristic and we also consider several dispatching rules. These dispatching rules are known from literature, but they are adjusted for the specific situation of an automated container terminal.

The remainder of this paper is organized as follows. First, we will describe the processes at a container terminal and the scheduling problem that arises. In Section 3, we will discuss how the static scheduling problem can be solved using a Beam Search algorithm and how this algorithm can be used in a dynamic context. In Section 4, we will report on our computational experiments with this approach. Next, in Section 5, we will discuss several dispatching rules known from literature and we will show that adjustments are necessary in order to avoid deadlock situations. Computational results of these dispatching rules will be discussed in Section 6. Finally, in Section 7 we will summarize our conclusions and give some directions for further research.

2 Problem setting

In this section, we give a detailed problem description. First, we will describe the environment in more detail. After that, we will elaborate on the loading and unloading operation of a vessel. Finally, we discuss some modeling issues.

2.1 Automated container terminals

A typical layout of an automated container terminal is given in Figure 1. From the figure it follows that the stack covers most of the area of the terminal. In the stack, the containers are temporarily stored as they change from one mode of transportation to another. For instance, a container may arrive at the terminal with a deep sea vessel and leave again on a truck, train, river barge or short sea vessel. The stack is divided into a number of stack lanes on which a dedicated stacking crane (ASC) is operating to retrieve and store containers. The AGV's are driving in a clock-wise loop. The transparent AGV's are empty, the shaded AGV's are loaded.

Whenever a container is to be loaded into the vessel, the proper ASC picks the container from the stack and transports it to the transferpoint at the seaside. At the transferpoint, the ASC loads the container on an empty AGV. Next, the AGV transports the container to the Quay Crane (QC), which lifts the container off the AGV and loads it into the vessel. Note that the QC is still manually operated. The handling of a container that is unloaded from the vessel is

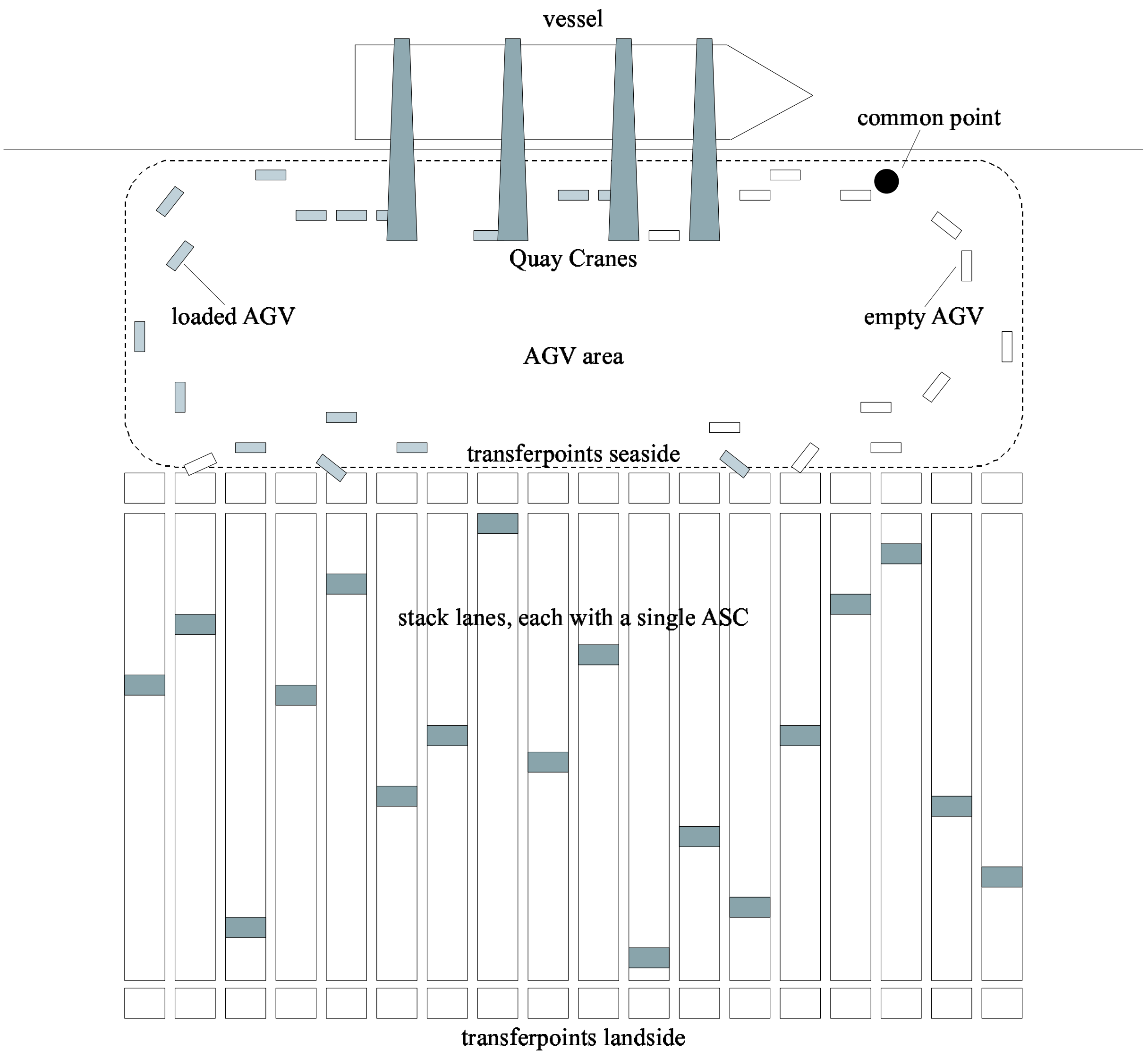

Figure 1: Typical layout of an automated container terminal

done in reverse order.

We assume that whenever an AGV drives from one location to another, it follows a predetermined path in the AGV area. Moreover, we assume that there is a common point which all AGV's pass after they have been unloaded by one of the QC's and drive towards the pickup location (ASC's) for their next container. These assumptions are satisfied in the circular layout as given in Figure 1 and also in many practical situations. For instance, at the automated container terminals in Rotterdam, the AGV's drive in a similar loop as in Figure 1. In case of more general layouts, such as layouts in which AGV's drive in both directions or layouts in which AGV's can take shortcuts directly after unloading at the QC and cross the middle area, we refer to the model as discussed in Meersmans *et al.* (2001).

2.2 Loading operation of a vessel

The containers are loaded into the ship according to a so-called stowage plan. The stowage plan is usually given by the shipping company and assigns each individual container to a specific position in the ship. This is done in such a way that the stability of the ship is maintained and the number of shifts, that is, the unnecessary unloading and reloading of containers at other ports, is minimized. For more details on stowage planning, see Avriel *et al.* (1998, 2000).

Modern container vessels have over 25 bays in which containers are loaded. The size of these vessels allows for the deployment of 3 to 6 QC's at the same time, where each QC handles a number of bays. The assignment of bays to QC's, and the order in which a QC handles the bays, is a problem in itself and is beyond the scope of this paper. The interested reader is referred to Daganzo (1989) and Peterkofsky & Daganzo (1990). Within a bay, the containers are loaded in a fixed order. Moreover, for reasons of visibility (the QC's are still manned), the positions at the waterside of the vessel are loaded first. So, since we know the order in which the QC handles the bays and since within a bay also the order in which the containers are loaded is fixed, we obtain a linear order of the containers to be loaded by the same QC.

Note that the unloading operation of a vessel is far less complex than the loading operation. This is because the unloaded containers can be randomly stored in the stack, that is, there is no such thing as a stowage plan that has to be respected. Therefore, we will only consider the loading of operation of a vessel.

As already mentioned in the introduction, the operating costs of seagoing vessels are very high, up to $1000 an hour for the current generation of vessels. These costs are likely to increase as even larger vessels are put into operation in the near future. Since seagoing vessels spend a major part of their time in ports, it is crucial to have the vessel loaded as fast as possible. Therefore, our goal is to minimize the loading time of the vessel, that is, to minimize the time at which the last QC has finished loading. Only then, the ship can depart.

2.3 Problem characteristics

In this subsection, we will discuss the main characteristics of the problem of scheduling the ASC's, AGV's and QC's so as to minimize the loading time in a static environment. As shown in Meersmans & Wagelmans (2001), this problem is NP-hard.

Blocking constraints
An important characteristic of the AGV's is that they are not able to load and unload containers themselves, i.e., a crane is always needed. This gives rise to blocking constraints. Once the ASC has picked up a container, it cannot advance to handle its next container until the proper (empty) AGV has arrived at the transferpoint and the container is loaded onto the AGV. Hence, the ASC is blocked by the AGV. Moreover, the AGV cannot advance to handle its next container until the QC has lifted the container off the AGV. So, the QC blocks the AGV.

Time-lags
Consider two containers i and j to be loaded by the same QC immediately after each other. We have that the AGV task related to container j cannot be finished before the QC has finished

loading container i. This is due to the blocking of the AGV by the QC. So, the loading sequence of the containers of a QC determines a similar order on the completion times of the related AGV tasks. This allows us to model the QC tasks implicitly by defining time-lags on the AGV tasks. Time lags are a generalization of precedence constraints and define general timing restrictions between start and/or completion times of tasks. So, for containers i and j, such that j is loaded immediately after i, we have that the difference in completion times of the corresponding AGV tasks is at least the handling time of container i by the QC. Note that the time-lags are chain-like, i.e., for each QC we have a linear order of the tasks (which is known in advance).

Fixed assignment of containers to ASC's
From the location of a container in the stack follows the ASC that will handle this container. Hence, each ASC has to handle a subset of all containers, where the elements of the subsets are known in advance, but the order in which they will be handled is not.

2.4 Dominant schedules

Recall that there is a common point that each AGV passes whenever it has delivered a container at a QC and drives back to the stack area (see Figure 1). Now suppose that when an AGV passes the common point, it is assigned its next container. Furthermore, suppose that this assignment is done according to a prespecified order of the containers, to which we will refer as an *assignment order,* or simply as an assignment.

The next theorem, which has been proven in Meersmans & Wagelmans (2001), states that for the static problem there is a dominant set of schedules in which the assignment order of the containers to the AGV's is consistent with the orders in which the ASC's handle the containers.

Theorem 2.1 *Consider an optimal schedule for the static problem. Let π_s denote the order in which ASC s handles its containers. Then there exists an optimal assignment order π of the containers to the AGV's, such that π_s is a suborder of π, for each ASC $s \in S$.*

Theorem 2.1 implies that the orders in which ASC's handle their containers can be derived from the assignment order of the containers to the AGV's, i.e., the assignment order π completely determines the schedule. So, we may enumerate over all possible assignment orders to obtain an optimal schedule. This observation is the basis of the Beam Search algorithm, which we will discuss in the next section.

3 Solving the integrated scheduling problem using Beam Search

In this section, we will first briefly present the Beam Search algorithm that was developed in Meersmans & Wagelmans (2001) for solving the integrated scheduling problem of terminal equipment in a static environment. Then we we will discuss how the Beam Search algorithm can be used in a dynamic context.

3.1 A Beam Search algorithm

Beam Search is a heuristic search technique that is closely related to Branch & Bound. Beam Search follows a breadth-first-search strategy for exploring the tree. However, instead of expanding all the nodes at a certain level of the tree, only a limited number of nodes are selected to be expanded further. The number of nodes selected is called the *beam width*. The selection of the most promising nodes that are kept for further branching, is done by using an evaluation function. Since large parts of the search tree are cut off in this way, the method runs very fast. For instance, in our Beam Search algorithm the number of generated nodes is $\mathcal{O}(bw \cdot n^2)$, where n is the number of containers and bw is the beam width, a parameter value that has to be chosen. At every level we generate $\mathcal{O}(bw \cdot n)$ new branches (nodes) and the complete tree consists of n levels (see below for a more detailed description of how the tree is constructed). Figure 2 shows a search tree that illustrates the Beam Search. From the root node, all branches are generated. At the first level, the best two nodes ($bw = 2$) are selected and further expanded. This procedure is repeated at level two, and so on, until we reach the leaves of the tree at level n.

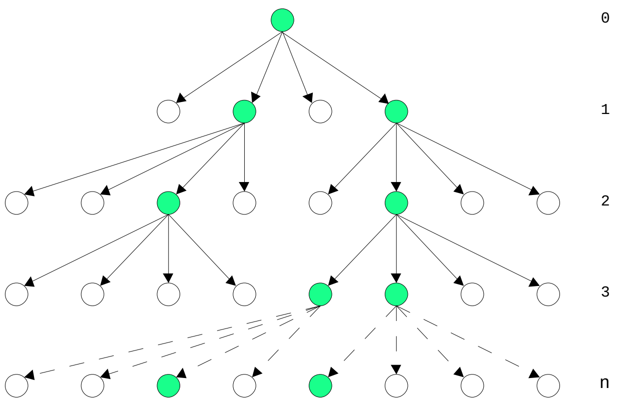

Figure 2: Beam Search algorithm with $bw = 2$

The selection of the nodes for further expansion is the crucial part within a Beam Search algorithm. However, a thorough evaluation can be computationally expensive. Therefore, a two–stage procedure is often used for the selection. In the first stage, the nodes are "filtered" using a simple evaluation function. After the filtering, there are only a limited number of nodes (referred to as the *filter width fw*) left. These nodes are evaluated again, now using a more detailed, time consuming evaluation function, which results in the set of nodes (bw) that are further expanded.

For the integrated scheduling of ASC's and AGV's, the complete search tree corresponds to an enumeration of all possible assignment orders. Recall that this suffices since, according to Theorem 2.1, both the schedule for the AGV's and ASC's can be represented completely by such an order. A node at level k of the tree represents a partial assignment of which the first k containers in the order are fixed.

The main ingredient of the Beam Search algorithm is the evaluation function that is used for selecting the most promising nodes for further expansion. In each node that is evaluated, we calculate a lower bound on the minimum loading time that will result if the partial assignment order would be expanded to a complete assignment order. The selection of the nodes is then done using these lower bounds. So, the *bw* nodes with the smallest lower bound are selected for further expansion. In case of ties, we select the node with the smallest upper bound, which is calculated as follows. Finish the partial schedule of the first k containers by adding the unscheduled containers in order of their *tail* t_i, which is defined as the handling time of the container on the QC plus the handling times of its successors on the QC, i.e.,

$$t_i = \sum_{l \succeq i} p_l^{qc} \tag{1}$$

The last major ingredient of the Beam Search algorithm is the so-called filter. In our implementation, the filtering is based on the tail t_i. That is, we only evaluate the lower bound of nodes for which the tail belongs to the *fw* largest.

Clearly, setting the beam and filter width requires some computational testing. However, as results in Meersmans & Wagelmans (2001) show, the solutions are not very sensitive to the specific parameter settings.

Computational tests showed that the Beam Search algorithm very effective. It produces, in a reasonable amount of time, solutions that usually are within 5 percent of the optimum, even for large problem instances. For more detailed information, we again refer to Meersmans & Wagelmans (2001).

3.2 Applying the Beam Search algorithm in a dynamic setting

In this subsection, we will discuss how the Beam Search algorithm can be applied in a dynamic setting, in which we have changing information or new information that becomes available as time goes by while the current schedule is executed. As a result, rescheduling may be advantageous since realized handling times may differ from the expected handling times that were used in constructing the schedule. Moreover, there may be additional containers that have to be incorporated in the schedule. In particular the latter will happen in a rolling planning horizon approach, which is what we will do. This means that at any point in time that scheduling decisions are taken, we only consider the containers that have to be handled relatively soon.

Note that for a given complete feasible schedule, differences between the expected handling times and the realizations do not result in an infeasible schedule. If we keep the order in which the containers are handled fixed, the schedule always remains feasible. The example in Figure 3 illustrates this. Given are four containers to be handled, one ASC and two AGV's, an we assume that the QC handling times are negligible. The upper Gantt chart represents the original schedule in which the ASC handles the containers 1 to 4 in this order and AGV 1 handles containers 1 and 3, and AGV 2 handles containers 2 and 4 (note the drive time d_3 and d_4 after the AGV's have completed the handling of their first container). The lower chart illustrates the schedule in which the handling time of containers 1 and 4 on the ASC are longer than expected and the handling of container 1 on the AGV is shorter than expected. However, the order of the containers is kept fixed. This still results in a feasible schedule, although the makespan is longer and there is idle time between the containers handled on the AGV's.

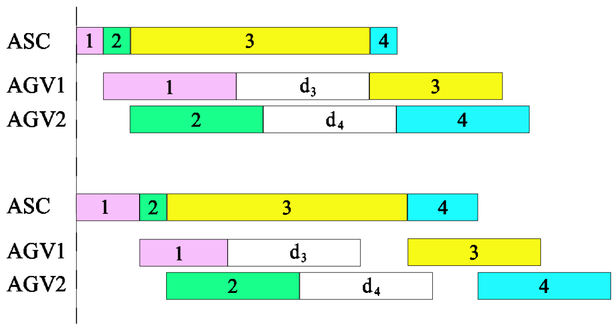

Figure 3: Gantt charts

Now suppose that containers are rescheduled, based on updated information about the handling time of the containers. Given the part of the schedule that has already been executed, it is not obvious that the Beam Search algorithm will find a feasible schedule. This is because the Beam Search does not exploit the search tree completely. Hence, we may have that the node which will finally result in a feasible solution, is cut off in an early stage. The following example illustrates this.

Example
Consider 2 QC's, 2 AGV's and 2 ASC's. The QC's load container 1,2 and 3,4 respectively. ASC1 has to handle containers 1 and 3, ASC2 handles containers 2,4. Suppose we have the initial schedule based on the assignment order $\pi = \{1, 3, 4, 2\}$. Whenever we execute this schedule, ASC 1 starts with container 1 and ASC 2 starts with container 4. Suppose we reschedule after container 1 is handled by the ASC. Moreover, we have that the tails t_i of the containers 2,3,4 equal 50, 100 and 49 respectively. In the Beam Search algorithm, the containers are initially sorted in order of non-increasing tail. Hence, initially we have the assignment order $\pi = \{3, 2, 4\}$. Now suppose we run the algorithm with filter width $fw = 2$ and beam width $bw = 1$. We then get the following search tree shown in Figure 4.

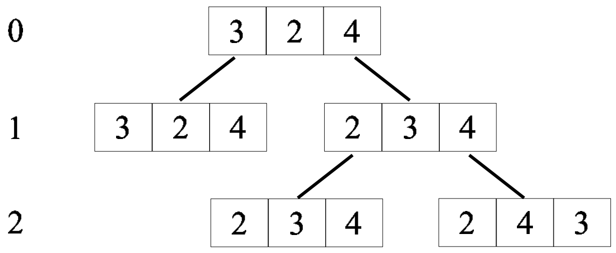

Figure 4: Search tree $fw = 2$, $bw = 1$

At level 0, we have the containers sorted according to their tail t_i. With a filter of 2, we generate the assignment orders on level 1. Now suppose the assignment order on the left leaf, has a larger lower bound than the assignment order of the right leaf. Since the beam width is only 1, we select the left leaf. This results then in the assignment orders on level 2, which are both infeasible, since ASC 2 has already started with container 4, but should in both cases deliver container 2 first.

Hence, the node representing the (obviously) feasible solution in which the assignment order is kept the same, may not be generated. So, rescheduling using the Beam Search algorithm should be done carefully in order to guarantee feasibility. Therefore, we propose the following procedure to deal with rescheduling.

Let π represent the assignment order of the AGV's (and ASC's), as defined in Theorem 2.1. Denote by i the container which is currently handled by some AGV or ASC, and which position in the assignment order π is largest. Whenever we reschedule, we take the positions of all containers up to i in the assignment order fixed. So, only the positions of the containers that succeed container i in the assignment order can be changed. Although this may seem to be a restrictive policy, it is necessary to guarantee feasibility at all times, since the Beam Search algorithm is used as a "black box".

4 Computational experiments with the Beam Search algorithm in a dynamic context

In this section, we will report on the computational experiments with the Beam Search algorithm in a dynamic context. In particular, we are interested in the following:

(a) The effects on the overall performance of the length of the planning horizon, i.e., the effects of taking more or less information into account. Although the running time of the Beam Search algorithm is polynomial in the number of containers to be scheduled, solving extremely large instances (about 1000 containers) can take quite some computation time. In a real time setting, or whenever rescheduling is done frequently, one may choose to restrict the planning horizon if the loss in performance is limited.

(b) The effect of the frequency of rescheduling on the overall performance. It is interesting to see whether taking new information frequently into account pays off, or whether it leads to a worse performance, since reacting on every bit of new information may give an unstable schedule. Moreover, it is clear that every time we reschedule, an amount of computation time is necessary. Also this should be taken into account.

(c) The effects of uncertain handling times of the containers. This uncertainty is mainly present at the QC's, since this is still a manual operation. Moreover, there is uncertainty in the driving times of the AGV's, since some congestion may appear near crossings of AGV tracks. This uncertainty may influence both the best planning horizon and the frequency of rescheduling. Obviously, taking a lot of information into account, that is, taking a long planning horizon, may work counterproductive whenever this information turns out to be uncertain. Also, the best rescheduling frequency may be influenced by the uncertainty in the handling times. Although rescheduling every time that new information becomes available may seem attractive, rescheduling too often may lead to "nervousness" and decrease again overall performance.

In order to investigate the performance of various algorithms for the scheduling of terminal equipment, a detailed simulation model has been developed in close cooperation with Europe Combined Terminals (ECT), the operator of the automated terminals in Rotterdam. For more details, we refer to Meersmans *et al.* (1999). In the experiments that were used to test the performance of the Beam Search algorithm, approximately 1000 containers were generated, to be loaded by four Quay Cranes. In order to guarantee reliable outcomes, over 150 of these loading operations were simulated.

In Table 1, we show the relative performance under various settings for both the planning horizon and the frequency of rescheduling, in a completely deterministic scenario. That is, we assume that the information about the handling times of the containers is perfectly known. As a reference point, we take for each run the schedule that was obtained by scheduling all containers of the loading operation at once at the beginning of the operation and keeping this schedule fixed. In the remainder of this paper, we will refer to this as the "static" version of the Beam Search algorithm. The numbers given in Table 1 represent the average percentage deviation in performance and in brackets, the standard deviation.

The planning horizon is set to 10, 20, 30, 40 and 50 containers per QC, which represents a workload ranging from about a quarter up to an hour. The rescheduling interval is taken to be 250, 500, 750 and 1000 seconds. All computations were done on a pentium PC 400 MHz. Computation times for the various settings of the planning horizon and the rescheduling interval varied from 1 to 5 minutes for constructing the initial schedule, and at most 2 minutes for rescheduling. The shorter computations times for rescheduling can be explained from the fact that that whenever we reschedule, a part of the schedule is already fixed (see Subsection 3.2). Scheduling all 1000 containers at once (the static version of the algorithm), led to computation times which were around half an hour. Note that the computation times are such that it is feasible to apply the Beam Search algorithm in real time, even for rescheduling intervals of 250 seconds.

rescheduling	planning horizon (containers/QC)				
interval (sec.)	10	20	30	40	50
1000	13.2 (6.4)	10.5 (6.7)	9.6 (6.2)	3.4 (3.8)	1.3 (3.8)
750	12.9 (6.7)	10.1 (6.4)	7.5 (4.8)	2.3 (3.9)	1.0 (3.7)
500	12.9 (6.8)	9.5 (6.4)	5.6 (3.9)	1.4 (4.0)	0.8 (3.7)
250	12.0 (6.6)	7.9 (5.2)	3.8 (3.1)	0.8 (3.8)	0.6 (3.5)

Table 1: Performance (in percentage deviation from makespan of static schedule) under different planning horizons / rescheduling intervals; deterministic scenario

The results from Table 1 are in line with our expectations; the longer the planning horizon and the higher the frequency of rescheduling, the better the performance. In all cases, the static version of the algorithm has the best average performance. (Note that if the Beam Search were an exact algorithm, the static version would always perform best.) Moreover, we may conclude from Table 1 that the length of the planning horizon is more important to achieve satisfactory performance than the frequency of rescheduling. Taking a short planning horizon of about 20 or 30 containers per QC, which represents about half an hour of workload, yields a relative low performance, even if we reschedule frequently. On the other hand, when the planning horizon is increased to about an hour of workload (40 to 50 containers per QC), the performance of the dynamic version of the Beam Search algorithm is on average very close to the performance of the static version, even for relatively large rescheduling intervals.

The results from Table 1 also show that the standard deviations are quite high, even if the planning horizon is large. Looking at individual instances, we observed that for 10 to 20 percent of the instances, rescheduling resulted in relatively bad schedules, compared to the schedules obtained by scheduling all containers at once. For the remainder of the instances, the schedules are of almost similar quality. Consider, for instance, the particular combination of a planning horizon of 50 containers per QC and rescheduling every 250 seconds. We found that in the worst case, the dynamic version of the algorithm performs about 9 percent worse than the static version. The other way around, the dynamic versions performed in the best case about 5 percent better than the static version (note again that the Beam Search algorithm is a heuristic procedure and does not necessarily give the optimal solution).

In order to investigate the effects of uncertainty in the handling times of containers, we introduce two stochastic scenarios. In the "low" stochastic scenario, we assume that both the handling times at the QC's and the AGV's are uniformly distributed within a range of plus/minus 10 percent of the average handling time. In the "high" stochastic scenario, this percentage is set to 20. The dynamic version of the Beam Search algorithm obviously uses the information of the containers that have already been handled. So, the realizations of the handling times of these containers are known. However, for the containers that are not handled yet, the schedule is based on the expected values of the handling times.

Again we compare the dynamic version of the Beam Search algorithm with the static version in which we schedule all containers beforehand, using expected handling times. The results are summarized in Table 2 and Table 3 and give the average difference in performance and between brackets, the standard deviation.

rescheduling	planning horizon (containers/QC)				
interval (sec.)	10	20	30	40	50
1000	12.9 (6.3)	10.3 (6.8)	9.5 (6.0)	3.5 (4.0)	1.4 (3.7)
750	12.6 (6.6)	9.7 (6.5)	7.2 (4.7)	2.5 (4.0)	1.1 (3.5)
500	12.9 (6.5)	9.3 (6.1)	5.6 (4.0)	1.4 (3.7)	0.9 (3.5)
250	11.6 (6.7)	7.4 (5.0)	3.8 (3.1)	0.9 (3.4)	0.8 (3.4)

Table 2: Performance (in percentage deviation from makespan of static schedule) under different planning horizons / rescheduling intervals; low stochastic scenario

From the results of Table 2 and Table 3, we may again conclude that the length of the planning horizon is most important. Even in a high stochastic environment, the dynamic version of the algorithm does not give better results than the static version. So, we may conclude that the information about the handling of future containers, although uncertain, is more important than the correct information about already handled containers.

Note that the schedule we compute in the static version of the algorithm is the same, whether we consider a deterministic or a stochastic case. From Tables 2 and 3 we may therefore conclude that such a schedule is quite robust. This is also illustrated in Table 4, which gives the average

rescheduling	planning horizon (containers/QC)									
interval (sec.)	10		20		30		40		50	
1000	12.5	(6.4)	10.1	(6.6)	9.2	(6.2)	3.4	(3.8)	1.4	(3.5)
750	12.4	(6.4)	9.7	(6.4)	6.9	(4.4)	2.4	(3.7)	1.3	(3.4)
500	12.2	(6.3)	8.9	(6.2)	5.4	(4.1)	1.5	(3.8)	1.0	(3.3)
250	11.3	(6.3)	7.1	(4.9)	3.5	(3.3)	1.1	(3.6)	0.9	(3.3)

Table 3: Performance (in percentage deviation from makespan of static schedule) under different planning horizons / rescheduling intervals; high stochastic scenario

increase of the makespan of the static schedules when these are evaluated in the stochastic scenarios. As we can see from the table, the performance does not deteriorate much when stochasticity is introduced in the handling times of the Quay Cranes and AGV's.

scenario	deviation from makespan in deterministic case	
	average (%)	stand. dev.
low	0.4	(0.3)
high	1.4	(0.6)

Table 4: Influence of stochastic handling times on overall performance of static schedule

5 Dispatching rules

The advantages of dispatching rules are obvious. By definition, these rules do not use complicated mathematical models to calculate the "best" assignment. Hence, they are easy to implement and therefore often used by practitioners. Moreover, these rules require only few information about the system. For instance, using a rule like Nearest Workstation First (Van der Meer (2000)), an idle AGV simply selects the nearest workstation to pick up its next load. So, especially in situations in which information is not available or uncertain, these rules seem to be an attractive alternative. In this section, we will compare the performance of various dispatching rules with each other and with the more complicated Beam Search algorithm.

5.1 Some well known dispatching rules

In this subsection, we will discuss several dispatching rules from the literature that are commonly used within the area of (AGV) scheduling. Early work within this area was done by Egbelu & Tanchoco (1984), who were among the first to investigate the performance of various dispatching rules for AGV's within Flexible Manufacturing Systems. Moreover, we mention Van der Meer

(2000) who investigated the performance of dispatching rules for various AGV systems, among others, an AGV system at an automated container terminal.

The dispatching rules we will consider are the following:

1. *Nearest Vehicle/Workstation First* (Van der Meer (2000)). Assigns an idle AGV to the nearest available load or, alternatively, a load is assigned to the nearest idle AGV.
2. *First Come First Served* (Egbelu & Tanchoco (1984); Van der Meer (2000)). Assigns the first idle AGV to the load that has been available for the longest time.
3. *Random Assignment* (Egbelu & Tanchoco (1984)). Assigns an idle AGV randomly to a load.
4. *Fixed Assignment of AGV's to QC's.* This rule is currently used at container terminals (see, for instance, Steenken (1992) who applies this rule for straddle carriers). Under this rule, each AGV will transport only containers destined for a certain QC.
5. *Most Work Remaining.* This rule is commonly used in machine scheduling. The general idea is that the next job that is scheduled on an idle machine is the job for which the remaining handling time is largest.
6. *Earliest Due Date.* Another rule that is commonly used in machine scheduling. Assigns to an idle machine the job that is most urgent with respect to its due-date.

The dispatching rules as given above, are usually evaluated on a number of performance criteria, for instance:

- Maximization of vehicle utilization
- Maximization of system throughput
- Minimization of queue lengths
- Minimization of (empty) driving distance
- Balanced workload

It is obvious, that the behavior of some dispatching rules favors one or more of the criteria as mentioned above. For instance, the Nearest Vehicle/Workstation First rule will likely result in short empty driving distances. However, for the specific situation of a container terminal, all the performance criteria as mentioned above are subordinate to minimizing the time required to load all containers, i.e., minimizing the makespan of the schedule.

Note that the dispatching rules as discussed above only consider the dispatching of the AGV's. As we will see in the next subsection, a straightforward implementation of the dispatching rules will inevitably lead to deadlock situations. Moreover, also a schedule for the ASC's has to be determined, which must be coordinated with the AGV dispatching. In Subsection 5.3 we will discuss how the dispatching rules can be implemented such that a feasible schedule for both the AGV's and ASC's is guaranteed.

5.2 Deadlocks

Although the dispatching rules as discussed in the previous section seem to be generally applicable, the specific setting of a container terminal does not allow for straightforward implementation of such a rule. Applying dispatching rules as presented above instead of solving the integrated scheduling problem by using the Beam Search or another algorithm calls for some caution. Because of the blocking constraints, and the resulting dependence between the ASC and AGV schedules, deadlocks may appear. A deadlock is a situation in which there is a total standstill of the system, due to circular waiting of the equipment. All AGV's and ASC's are occupied and cannot be released of their containers. ASC's wait for empty AGV's, which are not available, since the loaded AGV's wait for a container that should precede at the QC, but which cannot be transported since all AGV's are occupied. The following example illustrates such a situation.

Example
Consider a simplified situation in which we have a single QC, two AGV's and three ASC's. The QC has to handle containers 1, 2 and 3 in this order. Each container is located in a different stack lane. Moreover, the handling times of the containers on the ASC's are $p_1^{asc} = 3$, $p_2^{asc} = 2$, $p_3^{asc} = 1$. Since each ASC has exactly one container to handle, each ASC will start doing so at time $t = 0$. As a result, the ASC's are ready to transfer the containers to the AGV's at times $t_1 = 3$, $t_2 = 2$, $t_3 = 1$. Suppose the two AGV's become empty at the QC at the same time and assume the driving times to the stack lanes are negligible. Now using, for instance, an AGV dispatching rule like Nearest Vehicle/Workstation First (Van der Meer (2000)), which assigns an idle vehicle to the nearest available load, or alternatively, assigns a load to the nearest idle vehicle, leads to the assignment of containers 2 and 3 to the two AGV's. As a result, both AGV's cannot be released from their container since container 1 should arrive first at the QC. This deadlock situation is illustrated in Figure 5.

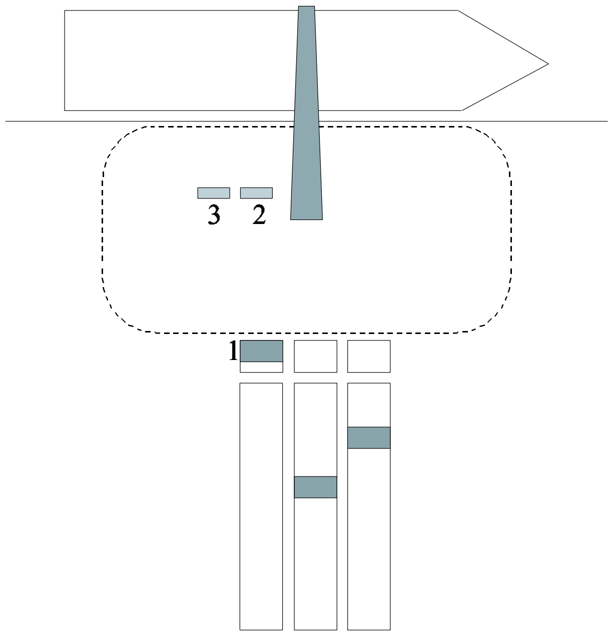

Figure 5: Example of a deadlock situation

Although the example of Figure 5 may seem somewhat artificial, it illustrates perfectly the complicated deadlocks that frequently occurred after we implemented the dispatching rules in our detailed simulation model of the container terminal (see Section 4).

In Flexible Manufacturing Systems, when deadlocks appear, intervention is usually possible by directing some vehicle to a special buffer area where the load is temporarily stored (see Egbelu & Tanchoco (1984)). Although intervention is also possible at an automated container terminal, this is not preferred since it leads to a (partial) shutdown of the terminal (all AGV's and ASC's are stopped). As a consequence, there is a great loss of performance. So, any dispatching rule that is implemented should ideally be such that deadlocks cannot appear. We will therefore adjust the dispatching rules as given in Subsection 5.1 in such a way that they always guarantee deadlock free schedules.

5.3 Deadlock free dispatching rules

In this subsection, we will discuss how the dispatching rules as mentioned in Subsection 5.1 can be modified in order to guarantee feasibility. The following result is used to achieve this.

Theorem 5.1 *Consider the integrated scheduling problem of ASC's, AGV's and QC's. For a schedule to be feasible, the following two conditions are sufficient:*

1. *The order of the containers on the ASC's and the AGV's coincide, i.e., for each ASC s, the order π_s of the containers is a suborder of π, the assignment order of the containers to the AGV's.*
2. *The order of the containers on the QC's and the AGV's coincide, i.e., the fixed order π_q of the containers on QC q, should be a suborder of π, for each QC q.*

Proof: We will prove the theorem by induction. Given that the first i containers in the assignment order π give a feasible schedule, we will show that adding the $i+1^{th}$ container also results in a feasible schedule.

Note that since the schedule for the first i containers is feasible, we have that all containers are completed and thus, all AGV's have a finite finish time. Take now the first idle AGV and assign this AGV to the $i+1^{th}$ container. Clearly, since the orders π_s are in accordance with π, the $i+1^{th}$ container can be completed on the ASC, since all its predecessors in π_s are completed (the schedule of the first i containers is feasible). Moreover, since the orders π_q are suborders of π, we also have that all predecessors of the $i+1^{th}$ container at the QC are completed. So, also the $i+1^{th}$ container can be completed on the QC and thus, we have a feasible schedule. □

Note that Theorem 5.1 only gives sufficient conditions for a schedule to be feasible. So, there may be schedules that do not satisfy these conditions, but which are still feasible. However, by their nature, dispatching rules are straightforward priority rules and therefore, we do not want to complicate them too much.

Theorem 5.1 gives us a guide to develop dispatching rules that guarantee a feasible solution. We will next discuss if and how we can modify the dispatching rules we discussed earlier. We will focus on the AGV assignment. The order of the containers on the ASC's is kept in accordance with this AGV assignment, i.e., we satisfy Condition 1 of Theorem 5.1.

1. *Nearest Vehicle/Workstation First.* This rule cannot be applied here. Recall the layout of Figure 1 in which all AGV's pass a common point after unloading. So, there is in fact only a single delivery location. Applying this rule will assign all AGV's to a container that is located in the nearest stacking lane. Obviously, this will lead to deadlock situations since this dispatching rule does not respect the order of the containers at the QC in any way.

2. *First Come First Served.* In order to determine which container has been available the longest time, we define a planning horizon for each QC consisting of k containers. All containers are pooled in a list of unscheduled containers which is initially sorted according to the time at which the containers are required at the QC (based on the expected handling times of the containers at the QC). After a container is handled by a QC, the $k+1^{th}$ container for this QC is made available. This container is then added last to the list of unscheduled containers. If an AGV becomes idle, the first container on the list of unscheduled containers is assigned to it. In this way, we ensure that the AGV is assigned to the container that was generated earliest. Note that by this strategy, the schedule will always respect Condition 2 of Theorem 5.1.

3. *Random Assignment.* To ensure the feasibility of this dispatching rule, we assign the empty AGV to the earliest required unscheduled container of a randomly chosen QC. In this way we ensure that Condition 2 is satisfied.

4. *Fixed Assignment of AGV's to QC's.* The implementation of this rule is quite obvious. Each AGV transports containers destined for only a single QC. An empty AGV is assigned to the unscheduled container which is required earliest at the QC.

5. *Most Work Remaining.* Crucial here is to determine which "container" has the most work remaining. We have implemented this rule by assigning the idle AGV to the unscheduled container which tail t_i is largest.

6. *Earliest Due Date.* To implement this rule, we take for each container the time at which the container should be available at the QC, such that there is no idle time for the QC. Given this time, we can calculate the time at which the AGV should leave the common point latest to load the container at the stack lane and drive to the proper QC in order to be just in time. We take this as the due date of a container. An empty AGV that arrives at the common point is then assigned to the container with the smallest due date. For two container i, j destined for the same QC, such that i precedes j, we should have that the due date of container i is smaller than the due date of j. This is obvious whenever we have a perfect loop layout. However, in case we only have a common point, we may have that the total handling time on the AGV of container j is larger than the total handling time of container i, and hence, the due date of j may be smaller than the due date of i (if i and j are in different stack lanes). If this is the case, we force the due date of j to be slightly larger than the due date of i. In this way, we ensure Condition 2 of Theorem 5.1. Note that each time a container is loaded by one of the QC's, we can update the due dates

of all the containers related to that QC, taking into account the most recent information. However, this has to be done carefully. For instance, a container of which the handling has already been started by the ASC, but not yet by the AGV, should not be updated. Otherwise, the due date of the container may increase, and as a result, the idle AGV may be assigned to another container from that stack lane, resulting in a deadlock situation. So, we should always guarantee that the AGV's handle they containers in accordance to the order of the containers on the ASC's (Condition 1 of Theorem 5.1)

6 Dispatching rules versus Beam Search

In this section, we will discuss our computational study to investigate the performance of the various dispatching rules discussed in Subsection 5.3. Moreover, we will compare the results of the dispatching rules with the results of the Beam Search algorithm, both in its static and its dynamic version.

Both the Beam Search algorithm and the dispatching rules were tested using the simulation model mentioned in Section 4. Moreover, the same loading operations (over 150) were generated as in Section 4. As a benchmark, we use the results obtained by applying the static Beam Search algorithm. The results of the dispatching rules are compared with this benchmark. We only evaluate the dispatching rules based on the time the last container is loaded onto the ship, i.e., the makespan of the schedule. Other criteria, like vehicle utilization and driving distance, as mentioned in Subsection 5.1, were not considered since they are all subordinate to the minimization of the makespan. Table 5 gives the average deviation of the makespan of the schedules generated by the dispatching rules (in percent), compared to the static version of the Beam Search algorithm. Between brackets, the standard deviations are given. Note that we repeat the results of the best dynamic Beam Search algorithm (see Tables 1 to 3).

dispatching rule	scenario					
	deterministic		low stochastic		high stochastic	
First Come First Served (10)	12.4	(5.7)	12.2	(5.5)	11.8	(5.4)
First Come First Served (50)	2.4	(1.9)	2.3	(2.0)	2.1	(2.1)
Most Work Remaining	2.4	(3.3)	2.2	(3.2)	1.9	(3.2)
Random Assignment	12.5	(4.6)	12.1	(4.6)	11.6	(4.6)
Fixed Assignment	7.1	(4.8)	7.1	(4.7)	7.0	(4.5)
Earliest Due Date	13.8	(6.8)	13.4	(6.5)	13.0	(6.4)
Beam Search (50, 250)	0.6	(3.5)	0.8	(3.4)	0.9	(3.3)

Table 5: Dispatching rules vs. Beam Search (performance in percentage deviation from makespan of static schedule)

The following remarks are in order:

1. The performance of the First Come First Served rule depends on the length of the planning

horizon that is taken into account. In Table 5 we give results for both a horizon of 10 and 50 containers per QC. The less containers are within the planning horizon, the more likely it is that an ASC becomes idle. The explains the relatively poor results for the horizon of 10 containers.

2. For the Fixed Assignment rule, the numbers in Table 5 are only based on cases in which the number of AGV's is a multiple of the number of QC's, since in the Fixed Assignment rule, each QC has a fixed number of AGV's assigned to it.

The results in Table 5 give rise to the following conclusions. First of all, the average performance of the Fixed Assignment rule, that is often used in practice, can be improved upon significantly. Furthermore, their is a lot a variation in the performance of the different dispatching rules. The dynamic version of the Beam Search algorithm performs best on average, although the Most Work Remaining rule and the First Come First Served rule with a large planning horizon come very close. These two rules take information about future containers into account. Therefore, their good performance confirms our conclusion from Section 4 that it is more important to take information about future containers into account than to react quickly to realizations of handling times. For instance, the Earliest Due Date rule is based on constantly updated information about when a container is required at the QC. Its performance, however, is the worst of all. Moreover, updating information should be done carefully, i.e., the conditions of Theorem 5.1 should be taken into account. From computational tests we learned that neglecting these conditions (for instance, by defining dispatching rules which are less strict) will often lead to deadlock situations. Note, however, that the conditions of Theorem 5.1 have not been proven to be necessary to obtain a feasible schedule. There may exist conditions that are less strict but still guarantee feasibility.

Another conclusion we can draw from Table 5 is that the dispatching rules tend to perform slightly better, relative to the Beam Search, when the stochasticity increases.

7 Conclusions and further research

In this paper, we considered the scheduling of container terminal handling equipment within a dynamic and stochastic context. We compared the performance of various algorithms under both deterministic and stochastic scenarios.

The integrated scheduling problems can be solved by using, for instance, a method like the Beam Search algorithm which was discussed in Section 3.1. This algorithm models the problem exactly and solves it using partial enumeration. A drawback of this method is that, although it runs in polynomial time (in the number of containers), the computation time becomes quite large for very large instances (1000 containers). Therefore, we also considered a dynamic version of the algorithm, which only takes a small number of containers into account within a rolling horizon. We investigated the performance of this dynamic version of the Beam Search algorithm and compared it to the performance of the static version of the algorithm, that solves the whole problem at once. In particular, we investigated the effects of the length of the planning horizon and the frequency of rescheduling were investigated in an extensive computational study. The results show that the length of the planning horizon is the most important factor for the performance, i.e., the longer the planning horizon the better the average performance. This result holds for both deterministic and stochastic scenarios. Hence, it seems that taking information about future containers into account is advantageous, even if this information is not completely reliable. This also means that the static version of the algorithm, which uses the longest possible planning horizon, performs best on average.

The second part of the paper considered various dispatching rules. It is said that such rules are very general and easy to implement. Therefore, they are often preferred by practitioners. However, we have shown that a straightforward implementation of such rules will lead to deadlock situations. Hence, we presented modified versions of such rules which do yield feasible schedules. Moreover, we compared the performance of these rules with the performance of the Beam Search algorithm we considered earlier. It is found that, on average, the Beam Search algorithm performs the best, but that some dispatching rules such as First Come First Served and Most Work Remaining come very close.

For practical situations, our results have the following implications. First of all, it does not seem to be a good idea to stick to the Fixed Assignment rule that is often used in practice. Given its relative performance and robustness we recommend to use the static version of the Beam Search algorithm. Note that all computations can be carried out in advance, so computation time need not be a bottleneck. In case unanticipated events, such as machine breakdowns, occur while the static schedule is being executed, rescheduling can always be done using the dynamic Beam Search algorithm with a planning horizon that is as long as possible. Alternatively, one may choose to use the modified First Come First Serve rule (with a long planning horizon) or the modified Most Work Remaining rule. Both may give slightly worse results, but they are much easier to implement than the Beam Search approach.

Finally we note that in Theorem 5.1 two sufficient conditions for the feasibility of a schedule were given. Future research may focus on determining necessary conditions or sufficient conditions that are less strict. If such conditions can be found, they may lead to the development of better dispatching rules.

References

AVRIEL, M., PENN, M. & SHPIRER, N. (2000). *Container ship stowage problem: complexity and connection to the coloring of circle graphs.* Discrete Applied Mathematics, 103:271–279.

AVRIEL, M., PENN, M., SHPIRER, N. & WITTEBOON, S. (1998). *Stowage planning for container ships to reduce the number of shifts.* Annals of Operations Research, 76:55–71.

DAGANZO, C.F. (1989). *The crane scheduling problem.* Transportation Research B, 23B:159–175.

EGBELU, P.J. & TANCHOCO, J.M.A. (1984). *Characterization of automated guided vehicle dispatching rules.* International Journal of Production Research, 22:359–374.

MEERSMANS, P.J.M., VAN HOESEL, C.P.M. & WAGELMANS, A.P.M. (2001). *Integrated scheduling of handling equipment at container terminals.* (forthcoming).

MEERSMANS, P.J.M., VIS, I.F.A., DE KOSTER, R. & DEKKER, R. (1999). *Famas-newcon: een model voor korte termijn stacking (in Dutch).* Technical Report EI-9942/A, Econometric Institute, Erasmus University Rotterdam.

MEERSMANS, P.J.M. & WAGELMANS, A.P.M. (2001). *Effective algorithms for integrated scheduling of handling equipment at automated container terminals.* Technical Report EI 2001–19, Econometric Institute, Erasmus University Rotterdam.

PETERKOFSKY, R.I. & DAGANZO, C.F. (1990). *A branch and bound solution method for the crane scheduling problem.* Transportation Research B, 24B:159–172.

STEENKEN, D. (1992). *Fahrwegoptimierung am containerterminal unter echtzeitbedingungen.* OR Spektrum, 14:161–168.

VAN DER MEER, J.R. (2000). *Operational control of internal transport.* Ph.D. thesis, Erasmus University Rotterdam.

Publications in the Report Series Research[*] in Management

ERIM Research Program: "Business Processes, Logistics and Information Systems"

2001

Bankruptcy Prediction with Rough Sets
Jan C. Bioch & Viara Popova
ERS-2001-11-LIS

Neural Networks for Target Selection in Direct Marketing
Rob Potharst, Uzay Kaymak & Wim Pijls
ERS-2001-14-LIS

An Inventory Model with Dependent Product Demands and Returns
Gudrun P. Kiesmüller & Erwin van der Laan
ERS-2001-16-LIS

Weighted Constraints in Fuzzy Optimization
U. Kaymak & J.M. Sousa
ERS-2001-19-LIS

Minimum Vehicle Fleet Size at a Container Terminal
Iris F.A. Vis, René de Koster & Martin W.P. Savelsbergh
ERS-2001-24-LIS

The algorithmic complexity of modular decompostion
Jan C. Bioch
ERS-2001-30-LIS

A Dynamic Approach to Vehicle Scheduling
Dennis Huisman, Richard Freling & Albert Wagelmans
ERS-2001- 35-LIS

Effective Algorithms for Integrated Scheduling of Handling Equipment at Automated Container Terminals
Patrick J.M. Meersmans & Albert Wagelmans
ERS-2001-36-LIS

Rostering at a Dutch Security Firm
Richard Freling, Nanda Piersma, Albert P.M. Wagelmans & Arjen van de Wetering
ERS-2001-37-LIS

Probabilistic and Statistical Fuzzy Set Foundations of Competitive Exception Learning
J. van den Berg, W.M. van den Bergh, U. Kaymak
ERS-2001-40-LIS

Design of closed loop supply chains: a production and return network for refrigerators
Harold Krikke, Jacqueline Bloemhof-Ruwaard & Luk N. Van Wassenhove
ERS-2001-45-LIS

---

[*] A complete overview of the ERIM Report Series Research in Management:
http://www.ers.erim.eur.nl

ERIM Research Programs:

LIS	Business Processes, Logistics and Information Systems
ORG	Organizing for Performance
MKT	Marketing
F&A	Finance and Accounting
STR	Strategy and Entrepreneurship