EFFECTIVE ALGORITHMS FOR INTEGRATED 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-2001-36-LIS
Publication	June 2001
Number of pages	26
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 of all the ERIM reports are also available on the ERIM website: www.erim.eur.nl

ERASMUS **R**ESEARCH **I**NSTITUTE OF **M**ANAGEMENT

REPORT SERIES

RESEARCH IN MANAGEMENT

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, where the objective is to minimize the makespan of the schedule. We present a Branch & Bound algorithm that uses various combinatorial lower bounds. Computational experiments show that this algorithm is able to produce optimal or near optimal schedules for instances of practical size in a reasonable time. We also develop a Beam Search heuristic that can be used to tackle very large problem instances. Our experiments show that for such instances the heuristic obtains close to optimal solutions in a reasonable time.	
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	Production Management
	R 4	Transportation Systems
	C 69	Mathematical methods and programming: other
European Business Schools Library Group (EBSLG)	85 A	Business General
	260 K	Logistics
	240 B	Information Systems Management
	250 A	Mathematics
	260 Q	Transportation
Gemeenschappelijke Onderwerpsontsluiting (GOO)		
Classification GOO	85.00	Bedrijfskunde, Organisatiekunde: algemeen
	85.34	Logistiek management
	85.20	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, scheduling, AGV's, Branch & Bound, Beam Search	

Effective algorithms for integrated scheduling of handling equipment at automated container terminals

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

May 31, 2001

Abstract

In this paper we consider the problem of integrated scheduling of various types of handling equipment at an automated container terminal, where the objective is to minimize the makespan of the schedule. We present a Branch & Bound algorithm that uses various combinatorial lower bounds. Computational experiments show that this algorithm is able to produce optimal or near optimal schedules for instances of practical size in a reasonable time. We also develop a Beam Search heuristic that can be used to tackle very large problem instances. Our experiments show that for such instances the heuristic obtains close to optimal solutions in a reasonable time.

keywords: integrated scheduling, Branch & Bound, Beam Search, container terminal

1 Introduction

Since its introduction in the mid-sixties, the container has rapidly taken over the market for intercontinental transport of general cargo. Big seagoing vessels transport containers between the continents. Ever since the sixties, the size of these vessels has increased enormously. Modern container vessels can carry up to 8000 TEU (Twenty foot Equivalent Unit). Because of the high operating costs (approximately $1000 per hour) and tight schedules of these vessels, unloading and loading in ports has to be done rapidly. This has lead to the development of specialized handling equipment. Nowadays, most terminals still operate manned equipment like straddle carriers or yard cranes and terminal trucks. In the port of Rotterdam, the Netherlands, however, several terminals use automated equipment like Automated Guided Vehicles (AGV's) and Automated Stacking Cranes (ASC's). Because the equipment is unmanned, and therefore driver's expertise is missing, efficient scheduling is crucial to achieve satisfactory performance of the container terminal. Moreover, due to the huge investment costs in terminal equipment, increased productivity of the terminal leads to large cost savings for the terminal operator.

Existing literature only deals with the scheduling of a single type of equipment, for instance the

*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.

scheduling of AGV's. However, due to the nature of the operations, the schedules of different types of equipment highly interact. In practice, the main loss of efficiency appears to be due to the fact that the schedules of, for instance, AGV's and ASC's are not coordinated. Hence, separated scheduling of equipment may lead to low overall performance or even to deadlock situations. To the best of our knowledge, none of the existing literature considers the integrated scheduling of various types of equipment at container terminals (see also Section 3). Therefore, in this paper, we propose a model that does deal with the integrated scheduling of all types of handling equipment and we will apply combinatorial optimization techniques to obtain optimal or close to optimal solutions of this model.

The paper is organised as follows. In the next section, we will give a detailed problem description, followed by a short overview of the existing literature in Section 3. In Section 4, we model the problem, discuss its complexity and derive some results that will be used in a Branch & Bound algorithm to be presented in Section 5. A Beam Search heuristic, that is based on the Branch & Bound algorithm, is discussed in Section 6. Computational results are given in Section 7, followed by conclusions and some directions for further research in Section 8.

2 Problem definition

In this section, we give a detailed problem description. First, we will describe the environment in more detail. Next, we will elaborate on the loading and unloading operation of a vessel.

2.1 Layout of the container terminal

At a container terminal, the containers are temporarily stored in the stack 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, riverbarge or short sea vessel. Obviously, there are peaks in the number of containers that arrive and depart at the terminal. For instance, whenever a seagoing vessel calls at the terminal, huge numbers of containers are unloaded and loaded in a short period of time. On the other hand, the arrival and departure process on the landside, for instance by truck or train, is much more stationary. Hence, the stack serves as a buffer to deal with the difference in supply and demand rates of the different modes of transportation.

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. 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 transparant AGV's are empty, the shaded AGV's are loaded. Note that after the AGV's are unloaded by one of the Quay Cranes (QC), they all pass a common point on their way to the next container to be picked up at a stack lane.

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 QC,

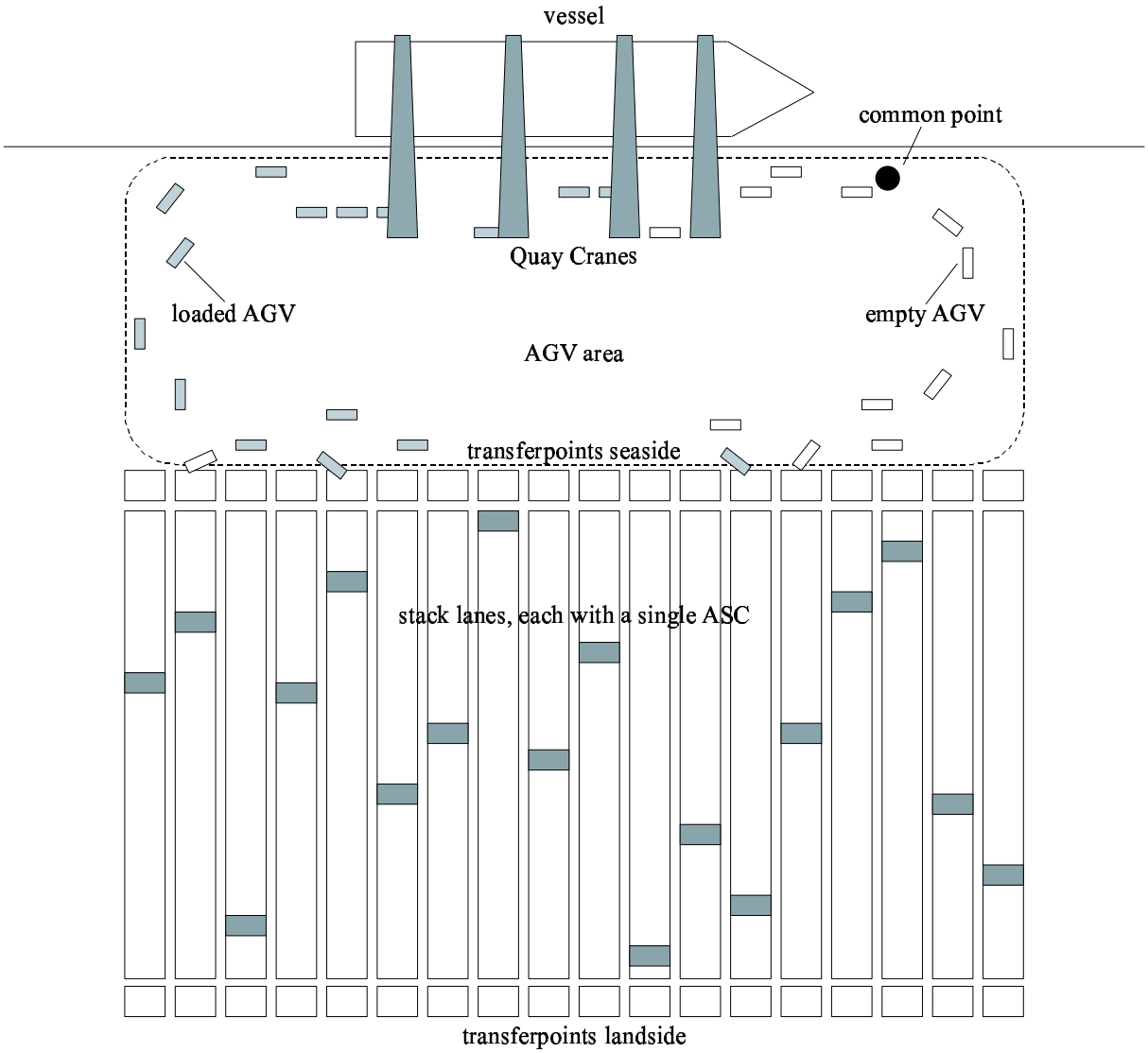

Figure 1: Typical layout of an automated container terminal

which lifts the container off the AGV and loads it into the vessel[1]. The handling of a container that is unloaded from the vessel is 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 whenever they are unloaded at the QC and drive towards the pickup location (ASC) for their next job. 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

[1]The handling of the container by the QC is still a manual operation

Meersmans *et al.* (forthcomming).

2.2 Loading and unloading 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 a single QC.

The unloading operation of a vessel is far less complex than the loading operation. Per bay, the containers are unloaded by the QC's and transported by AGV's to a randomly chosen stack lane. Within a stack lane, containers can be stored arbitrarily. For instance, heavy weight containers can be stored on top of light weight containers. Hence, the order in which the containers arrive at a stack lane is not relevant. Note that because of this, all containers that are unloaded are considered to be more or less the same. In case of a loading container however, weight, destination and other characteristics do matter, since the stowage plan has to be respected. This makes the operation more complex since the containers have to be delivered in the right order to the QC's. Therefore, we will consider only the loading of containers in this paper, since this is the most critical and complex operation in the handling process 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.

3 Related work

Within the area of container handling, most work concentrates on conventional, that is, manned handling systems. Moreover, all existing literature considers the optimization of a single type of handling equipment, like the scheduling of transfer cranes or the scheduling of AGV's. We will now discuss the most important references.

Kim & Kim (1999*a*) consider the optimization of the routing of a single transfer crane, comparable to the ASC we consider. Such a transfer cranes operates on a stacking block, which is again divided into a number of bays. The stacking blocks can be compared to the stack lanes we consider in this paper. The transfer crane picks the containers from the stack and loads them onto yard trucks which transport the containers to the quay cranes. The containers to be retrieved by the transfer crane are divided into a number of categories. The categories are based on size, weight, destination port, etcetera. Each bay is assumed to be dedicated to a category, i.e., it only contains a certain number of containers of the same category. Several bays can contain containers of the same category. For the transfer crane, the work schedule is given, that is, the sequence in which containers of a certain category have to be picked is given. The problem is then to determine the sequence in which the transfer crane visits the bays and the number of containers that is picked from a bay each time. Obviously, this has to be done in such way that the work schedule is satisfied. The problem is formulated as a mixed integer program. The special structure of this formulation is exploited in an optimization algorithm that uses dynamic programming. In a related paper by Kim & Kim (1999*b*), a Beam Search algorithm (see also Section 6) is used to solve the model.

Winter (1999) considers a terminal using straddle carriers. Unlike AGV's, straddle carriers are able to load and unload containers themselves. As a result, the straddle carriers are used for both the retrieval of containers from the stack and the transport to the QC's. The paper discusses the problem of combining the stowage planning of the vessel, with the scheduling of the straddle carriers. This is done as follows. The containers that are to be loaded are divided into a number of categories, based on size, weight, destination port, etcetera. For each QC a sequence of categories is then given. Hence, there is still freedom with respect to the choice of the specific container that is delivered, provided that the container is of the proper category. This additional freedom allows for the optimization of the movements of the straddle carriers.

A problem similar to the scheduling of a single ASC is the scheduling of a stacker crane in an automated warehouse. This problem is considered by Ascheuer *et al.* (1999). In particular, they consider an on-line setting, where a stacking crane has to perform several transportation tasks (storage, retrieval, etc.). The crane is able to travel both in horizontal and vertical direction. The problem is modeled as an Asymmetric Traveling Salesman Problem (ATSP) and both heuristics and an exact Branch & Cut algorithm are used to solve it. Related work of Ascheuer *et al.* (2000*a*,*b*) deals with polyhedral approaches to the ATSP and extensions, like the ATSP with precedence constraints and time windows.

The scheduling of AGV's is considered in Chen *et al.* (1997) and Kim & Bae (1999). Chen *et al.* (1997) consider the dispatching problem of AGV's in an automated container terminal. To simplify the analysis, they assume that yard cranes are always available to load or unload the AGV's. For the case of a single QC, with both loading and unloading operations, they show that a greedy algorithm is optimal. For the case of multiple QC's, they use the same algorithm as a heuristic. The algorithm is tested within a simulation model. In order to investigate the effectiveness and robustness of the algorithm, a queueing model is used.

Another approach to the scheduling of AGV's is given by Kim & Bae (1999). For each container to be transported, they introduce event times which have to be met. These event times define the exact pickup and delivery times of each container. For the case of a single QC and given the event times that have to be met, they reduce the dispatching problem of AGV's to an assignment

problem. For the case in which the event times cannot be met, a heuristic is developed.

Vis *et al.* (2001) consider the problem of determining the number of AGV's required at an semi-automated container terminal. They describe a model and give a minimum flow algorithm to solve the case in which containers are available for transport at given time instants. The traffic control of an AGV system at an automated container terminal is discussed by Evers & Koppers (1996). A distributed control system is proposed that uses a hierarchical system of semaphores, which roughly speaking, act as a kind of traffic lights.

Within Flexible Manufacturing Systems (FMS), AGV's are also used. Most of the literature considers the design aspects of such a system, for example Johnson & Brandeau (1993) and Thonemann & Brandeau (1997), or the performance of dispatching rules for the AGV's, see Egbelu & Tanchoco (1984) and Van der Meer (2000). Bilge & Ulusoy (1995) consider the simultaneous scheduling of machines and automated guided vehicles in an FMS. They propose a nonlinear mixed integer programming formulation and solve the problem heuristically using an iterative procedure and sliding time windows. However, they assume sufficient input and output buffer space at each machine and availability of (un)loading devices.

As already mentioned before, all literature considers the scheduling of only a single type of equipment. However, most authors do consider a terminal with both stacking cranes and transport vehicles. Obviously, the schedules of these cranes and vehicles highly interact since the crane (vehicle) cannot start its next job until the vehicle (crane) is ready. Hence, optimizing only one type of equipment does not necessarily lead to an improvement of the overall performance of the terminal. In the worst case, it can even lead to deadlock situations. Therefore, we propose a model that deals with the integrated scheduling of the handling equipment, allowing for the optimization of the overall performance of the container terminal. In the next section, we will discuss this integrated scheduling model.

4 Modeling and complexity

In this section we will model the scheduling problem as described in Section 2. We will discuss the complexity of the problem and we will derive the main result on which we will base the Branch & Bound algorithm (see Section 5). However, we will first discuss some modeling issues.

The loading of a single container corresponds to three jobs, which are carried out by different types of equipment; the ASC, the AGV and finally, the QC. Our objective is to minimize the time at which the last QC finishes loading. More formally, we want to design a schedule of which the makespan (denoted by C_{max}) is minimal. Now let n denote the number of containers to be loaded, and let Q, S and M denote the set of QC's, ASC's and AGV's, respectively. In the remainder of this paper, we will often refer to a container that has to be loaded as a job. Hence, each job consists of three operations and we introduce the parameters $p_i^{asc}, p_i^{agv}, p_i^{qc}$ to represent the processing times of job i on the ASC, AGV and QC respectively. The processing time on the ASC depends on the position of the container in the stack and the handling speed of the ASC for the different operations (lifting, driving, positioning). The processing and setup times on the AGV depend on the stack lane in which the container is stored and the position of the QC which will handle the container. Finally, the handling time of a container by the QC depends

on the position in the bay of the ship to which the container was assigned in the stowage plan. We assume that all processing and driving times are deterministic and that preemption is not allowed.

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 to perform these operations. This gives rise to blocking constraints. Once the ASC has picked up a container, it cannot advance to its next job until an 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 its next job until the QC has lifted the container off the AGV. So, the QC blocks the AGV.

Since the QC's block the AGV's and preemption is not allowed, the loading sequence of a QC determines a similar order on the completion times of the corresponding AGV jobs. This allows us to model the QC's implicitly by defining time lags on the AGV jobs. Time lags are a generalization of precedence constraints and define general timing restrictions between start and/or completion times of jobs. So if container 2 is immediately handled after container 1 by the same QC, the AGV job related to container 2 cannot be completed until the time that the QC has finished loading container 1. Hence the difference between the unloading times of the AGV's transporting containers 1 and 2, is at least equal to the handling time of container 1 by the QC. Note that the time lags are chain-like, i.e., for each QC we have a linear order of the jobs (see Section 2).

The location of a container in the stack implies which ASC will handle it. Hence, each ASC has to handle a subset of all containers, which is known in advance (see Section 2).

The complexity of this problem is stated in the following theorem:

Theorem 4.1 *The integrated scheduling problem as described above is NP-hard.*

Proof: See Appendix A. □

Theorem 4.1 does not give us much hope for finding a polynomial time algorithm. Therefore, we will develop a Branch & Bound algorithm in the next section. This algorithm is based on a result that essentially states that a schedule is completely determined by the order in which of the jobs are assigned to the AGV's. Hence, enumerating over all possible orders will provide us with the optimal solution.

First, we observe the following:

Observation 4.1 *We may restrict ourselves to schedules in which waiting times of the AGV's only occur due to the fact that the proper ASC or QC is not ready to load or unload a container.*

Now recall that there is a common point that each AGV passes whenever it has delivered a container at the QC and drives back to the stack area. Now suppose that when an AGV passes the common point, it is assigned its next job. Furthermore, suppose that this assignment is done according to a prespecified order of the jobs, to which we will refer as an assignment order,

or simply as an assignment.

Next, we will show that we may restrict ourselves to schedules for which the assignment order is consistent with the orders in which the ASC's handle the jobs.

Theorem 4.2 *Consider an optimal schedule. Let π_s denote the order in which ASC s handles its jobs. Then there exists an optimal assignment order π of the jobs to the AGV's, such that π_s is a suborder of π, for each ASC $s \in S$.*

Proof: We will prove this theorem by contradiction. Suppose we have an optimal assignment order π that is not consistent with some π_s. Hence, there is a pair of jobs i, j such that i precedes j in π_s and j precedes i in π. Next, we will show that reversing the jobs i and j in π, will give a schedule with the same makespan.

Let c_i and c_j denote the completion times of jobs i and j on the ASC. We have $c_i < c_j$ since i is scheduled before j on the ASC. Due to the blocking constraints, we have that at time c_i, the AGV transporting container i is available at the ASC. Moreover, since j precedes i in π, we have that the ASC assigned to container j arrived at the ASC not later than the AGV assigned to container i. Hence, it is also is available at time c_i. Now consider the same assignment order, except that jobs i and j swap positions. It is obvious that both AGV's are available again at the ASC at time c_i. Hence, the completion times of jobs i and j on the ASC can be kept the same. Keeping also the remaining part of the schedule fixed, we get a schedule with the same makespan. Repeating this argument for all jobs i, j for which the orders π and π_s do not correspond, we get the required result. □

Theorem 4.2 implies that the orders in which ASC's handle their jobs can be derived from the assignment order. Because of Observation 4.1, we then have sufficient information to completely specify a schedule. Hence, we may enumerate over all possible assignment orders to obtain the optimal schedule. In order to speed up the enumeration, we will develop a Branch & Bound algorithm which is discussed in Section 5.

Now define a partial assignment order π' as an assignment order that only specifies the assignment of the first k jobs (the other jobs are not assigned). Given this partial assignment order, we can then determine the completion times of the jobs on the different types of equipment. Each job that is started on an ASC, is also completed, given that there is an empty AGV available. So, if the k^{th} job is assigned to an AGV, this also means that this AGV is available and thus, the corresponding ASC job can be completed. For the QC's we have that every job which is started is also completed. Note however, that this does not necessarily mean that all k jobs are completed on the QC's. Since the QC's handle the jobs in a fixed order, it may happen that a job that is started on an AGV, cannot be started on the QC, since one of its predecessors on the QC is not available yet. Consequently, this implies that a job can be assigned to an AGV, although it cannot be finished (given the partial assignment of only the first k jobs). However, it may still be possible that this partial schedule can be completed such that it results in an overall feasible schedule. This brings us to the following definition of a feasible partial assignment order.

Definition 4.1 *Let π' be a partial assignment order of the first k jobs. Then this partial assignment order is called feasible if at least one AGV has a finite finish time in the corresponding partial schedule.*

Clearly, if for all AGV's the assigned jobs have to wait for a predecessor at the QC, this predecessor can never be transported to the QC, since there is no empty AGV. As a result, the partial schedule is can not be extended to a complete feasible schedule if no AGV has finite finish time. However,it will be shown in Subsection 5.4 that if at least one AGV becomes available again, it is always possible to extend the partial schedule to a complete schedule in which all jobs are carried out.

5 A Branch & Bound algorithm

Because of Theorem 4.2, it is possible to solve the scheduling problem by enumerating over all possible assignment orders π. In this section, we will present a Branch & Bound algorithm that implicitly enumerates these orders. We will now discuss the main ingredients of the Branch & Bound algorithm, that is, the branching rule, search strategy and lower and upper bound procedures.

5.1 Representation and branching rule

The nodes in the Branch & Bound tree represent a (partial) assignment order. At level $k < n$ in the tree, the partial assignment order specifies an assignment of the first k jobs. If this partial assignment is feasible (see Definition 4.1), we can determine the finish times of the QC's and ASC's and of at least one AGV.

The branching rule is then simply as follows. For each subproblem with the first k jobs fixed, we introduce branches by determining which job is assigned as the $(k+1)^{th}$. Since we generate a branch for each possible candidate, we obtain $n-k$ branches. Note that by branching in this way, the number of descendants per node is large in the upper part of the tree (whenever only the assignment order of a few jobs has already been fixed) and small in the lower part of the tree. In the next subsections, we will explicitly exploit the structure of the subproblem when calculating lower bounds.

5.2 Search strategy and global lowerbound

The search strategy chosen is depth-first-search. The advantage of this strategy is that the size of the tree that has to be stored in the memory is very limited, compared to, for instance, a best-first or breath-first strategy.

The algorithm is run several times in a row, each time allowing for a smaller tolerance (deviation from optimality). Hence, we try to find the optimal solution by successive approximation. The approximation is done as follows. Initially, we allow for a solution value which is 25 percent off

the optimum. So, given a best feasible solution found so far, each node with a lower bound that is within 25 percent of the value this solution so far is fathomed. In general, however, the gap between the current fathomed node and the best solution so far, will be less than 25 percent. So, during the search we keep track of the largest solution value gap between any fathomed node and the best solution found so far. Note that if the best solution so far is updated, we can also update the value of the largest gap.

After the run has finished, we calculate a global lower bound based on the value of the best solution found and the largest gap of any fathomed note. Next, we restart the algorithm using the value of the solution found in the previous run as an upper bound. Moreover, we set the tolerance equal to the largest gap found in the previous run, minus 1 percent. If after several runs, the tolerance falls under 1 percent, we run the algorithm using zero tolerance to obtain the optimal solution. Computational tests show that this strategy provides us quickly with good feasible solutions, which in turn allow for cutting off large parts of the search tree. Whereas in case of running the algorithm only once (with zero tolerance), we sometimes end up spending a lot of time in unpromising parts of the tree, trying to slightly improve a solution that turns out to be far from optimal.

Another advantage of this solution approach is that it allows for applying the Branch & Bound algorithm as a heuristic. Even if an optimal solution is hard to find, successively running the algorithm, each time with a smaller tolerance, provides us with a solution that is guaranteed to be within a known percentage from optimality. As we will see in Subsection 5.5, the Branch & Bound algorithm is stopped if the number of nodes becomes too large. If the current run of the algorithm is not completed, we use the lower bound from the previous run to calculate a bound on the gap between the best solution at termination and the optimum.

5.3 Combinatorial lower bounds

In this subsection we will discuss several combinatorial lower bounds that are used within the Branch & Bound algorithm. These lower bounds are all based on considering subproblems that can be solved efficiently. During the Branch & Bound algorithm, these lower bounds will be calculated in the order in which they are presented in the subsections. This order corresponds to increasing computational effort. Obviously, whenever the value of a lower bound is larger than the best solution found, we can fathom this node. After the calculation of each lower bound, we make this check. Hence, if the node can be fathomed, we do not have to calculate the other, more time consuming lower bounds. Computational tests show that of all fathomed nodes, 5 to 10 percent was fathomed due to the first lower bound procedure. About 40 percent was fathomed after calculating the second lower bound and the remaining part of the nodes was fathomed due to the third lower bound.

5.3.1 Quay Crane lower bound

An obvious lower bound is based on considering the containers to be scheduled on the same QC. Suppose we have a partial assignment that specifies the first k jobs. From the order of these jobs, we obtain for each QC the last job that it handles and the completion time of this job (see also Section 4). Denote by i_q the last job handled by QC q and let f_q be the corresponding

completion time on the QC of this job. Moreover, let $j \succ i$ represent the precedence relations on the QC, that is, job j succeeds job i. Then we can calculate for each QC a lower bound on the makespan C_{max} as follows:

$$\max_{q=1,2,\ldots,|Q|} \geq f_q + \sum_{l \succ i_q} p_l^{qc} \tag{1}$$

5.3.2 One Machine Scheduling lower bound

We now discuss a combinatorial lower bound based on a relaxation of a one machine scheduling problem. Again, we assume that we have a feasible partial assignment of the first k jobs. From this partial assignment, it follows at which time the ASC's complete their last job (in the partial schedule). Denote the finish time of ASC s by f_s.

For the remaining jobs to be scheduled on ASC s, we can now relax the problem by assuming that there is unlimited AGV capacity, i.e., the blocking property is ignored. Moreover, the time lags between the completion times of the jobs on the AGV's are omitted. As a consequence, the remaining jobs will be processed on the ASC with no idle time.

Now let $1, \ldots, n_s$ be the set of remaining jobs to be scheduled on ASC s. Then for each ASC s, we can calculate the following lower bound on the total makespan.

Theorem 5.1 *Let f_s denote the completion time of the last scheduled job on ASC s. Moreover, w.l.o.g. assume that the remaining jobs $1 \ldots n_s$ on ASC s are numbered in order of non-increasing tail, whhere the tail of job i is defined as:*

$$t_i^{asc} := p_i^{agv} + \sum_{l \succeq i} p_l^{qc} \tag{2}$$

Then a lower bound on C_{max} is the following:

$$\max_{s=1,2,\ldots,|S|} \max_{i=1,2,\ldots,n_s} \{f_s + \sum_{l=1}^{i} p_l^{asc} + t_i^{asc}\} \tag{3}$$

Proof: Equation (3) follows from the fact that, given unlimited AGV capacity, it is optimal to schedule the jobs on the ASC in order of non-increasing tail. To see that this is indeed optimal, suppose that there is a pair of jobs i, j such that $t_i^{asc} > t_j^{asc}$ and j is scheduled immediately before i in an optimal solution of the relaxed problem. Let PP_j denote the total processing time on the ASC of all the jobs in $\{1, 2, \ldots, n_s\}$ that are scheduled before j in this optimal solution. Since $t_i^{asc} > t_j^{asc}$, we have

$$f_s + PP_j + p_j^{asc} + p_i^{asc} + t_i^{asc} > f_s + +PP_j + p_j^{asc} + t_j^{asc} \tag{4}$$

Hence, the makespan of the optimal solution is at least equal to the left hand side of this inequality. It is easily verified that reversing jobs i and j does not increase the makespan, since

$$f_s + PP_j + p_j^{asc} + p_i^{asc} + t_i^{asc} > \max\{f_s + PP_j + p_i^{asc} + t_i^{asc}, f_s + PP_j + p_i^{asc} + p_j^{asc} + t_j^{asc}\} \tag{5}$$

In case there is another pair of consecutive jobs that are not in scheduled in the order of non-increasing tails, we can repeat the argument. □

Note that the bound (3) is calculated for all unassigned jobs, since we consider all ASC's and for each ASC we consider all jobs to be scheduled on this ASC.

5.3.3 Parallel Machine Scheduling lower bound

The following lower bound is based on considering the AGV's as a set of identical parallel machines. Assume that there is unlimited ASC and QC capacity. So, the ASC is always ready to load a container onto an AGV and the QC is always ready to unload a container from an AGV. Hence, the blocking property can be omitted. As a result, we can add the driving times from the common point to the ASC and the driving time from the QC to the common point to the processing time on the AGV. So, for each job we redefine the processing time on the AGV as:

$$\tilde{p}_i := d_{i1}^{agv} + p_i^{agv} + d_{i2}^{agv} \tag{6}$$

where d_{i1}^{agv} and d_{i2}^{agv} are the driving times from the common point to the ASC and from the QC to the common point, respectively. Note that in case the AGV's are driving in a loop-layout, we have $\tilde{p}_i = \tilde{p}$ for all jobs i, which we will assume in the analysis below. For any other layout with a common point for which the redefined processing times are not equal, we can perform the same analysis by defining

$$\tilde{p}_i := \min_{j=1,2,\ldots,n} \{d_{j1}^{agv} + p_j^{agv} + d_{j2}^{agv}\} \tag{7}$$

Similar to the tail defined in the previous subsection by equation (2) we define for each job a tail t_i^{agv} as follows:

$$t_i^{agv} = \sum_{l \succeq i} p_l^{qc} \tag{8}$$

A lower bound on C_{max} is now the following:

$$\min \max_{1,2,\ldots,n} \{c_i + t_i^{agv}\} \tag{9}$$

where c_i is the completion time of job i on its AGV. Since all AGV processing times are equal for all jobs, it is easily seen that the jobs should be processed in order of non-increasing tail in order to obtain the minimum in (9). Note that this means that we implicitly respect the precedence constraints among the jobs belonging to the same QC. This is because for every pair of jobs i, j such that $i \prec j$, we have that $t_i^{asc} > t_j^{asc}$. However, we do not necessarily respect the time lags among these jobs. That is, for two consecutive jobs i, j on a QC, we do not necessarily have that job j is completed on the AGV at least p_i^{qc} time later than job i is completed.

In general, we have a partial feasible assignment order (Definition 4.1) and a corresponding partial schedule when the above lower bound is calculated in a node of the tree. The idea is to process the remaining (unassigned) jobs on the AGV's in the order indicated above as soon as AGV's become available. Recall, however, that there may be AGV's for which the completion time of their current job cannot be determined yet. Therefore, we derive a lower bound on the earliest possible time that these AGV's become available again. Obviously, we would like this lower bound to be as large as possible. Hence, we calculate the following two bounds and select the largest:

1. A QC lower bound, similar to the bound as discussed in Subsection 5.3.1. Let i be the job handled by the AGV and let j be the last job handled by the proper QC. Given the finish time of the QC, we can calculate the earliest time at which the QC is available to handle job i by summing up all processing times on the QC of jobs $j+1, \ldots, i-1$. This gives us a lower bound on the time at which the AGV becomes available again.

2. A lower bound based on the remaining processing time of the predecessors on the ASC, similar to the bound discussed in Subsection 5.3.2. Let i be again the job handled by the AGV and let j be the last job handled by the QC. Now define H_s to be the set of jobs h that still have to be handled by ASC s and for which we have that $j \prec h \prec i$ on the QC. Clearly, if we want to have the AGV of job i to be released as soon as possible, the ASC should first handle all the jobs in set H_s, in the same order in which the jobs will be handled by the QC. Hence, we can calculate the following lower bound on the finish time f_i of the AGV to which job i is assigned:

$$f_i = \max_{s \in S} \max_{h \in H_s} \left(\sum_{l \in H_s:\ l \preceq h} p_k^{asc} + \sum_{l \succeq h}^{l \prec i} p_l^{qc} \right) \tag{10}$$

5.4 Upper bound heuristic

In each node of the branching tree, the partial assignment of the first k jobs is completed by applying the following heuristic. The unassigned jobs are sorted in order of non-increasing tail t_i^{agv}, as given by (8) and are added to the partial assignment in this order. As a result, we obtain a complete assignment. Now suppose that the partial assignment is feasible, as defined in Definition 4.1. Then we will show below that the complete assignment obtained by the heuristic is feasible in the sense that all jobs will be completed. Clearly, the objective value of any feasible solution is an upper bound on the makespan.

Theorem 5.2 *Consider any partial assignment of the first $k < n$ jobs which is feasible according to Definition 4.1. Completing the assignment by adding the unassigned jobs in order of non-increasing tails t_i^{agv} results in a feasible assignment.*

Proof: Since we complete the partial, feasible assignment by adding the unassigned jobs in order of non-increasing tails, it is sufficient to show that adding the first job in this way, gives us an extended partial assignment that is also feasible or, if $k = n-1$, a complete assignment that is feasible. Note that the latter will be the case if at least one AGV has a finite finish time,

since this means that all containers are picked up from their ASC and transported to their QC. Hence, in both cases it suffices to show that after adding the job, again at least one AGV will have a finite finish time.

First, observe the following. By definition of the tails t_i^{agv} (8), the following property is satisfied. For each pair of jobs i, j such that i precedes job j at a QC, we have:

$$t_i^{agv} > t_j^{agv} \tag{11}$$

Hence, the job to be added to the partial assignment is a job for which all predecessors on its QC have already been assigned. Moreover, these predecessors are all completed in the partial schedule, because otherwise it would not be possible that one of the AGV's has a finite finish time. Hence, the added job is handled by an AGV that originally had finite finish time. This AGV picks up the container (which is possible since all ASC's have finite finish time in the original partial schedule) and transports it to the QC, which processes it immediately. As a result, the AGV has a finite finish time again. □

5.5 Termination of the algorithm

Clearly, the Branch & Bound algorithm is terminated whenever the complete search tree has been investigated using zero tolerance and thus, the optimal solution is found. However, for large problem instances, the computation time might become excessively large. Therefore, we restrict the number of nodes that are generated in the tree to $1 \cdot 10^5$. Although this number may seem high, an efficient implementation of the algorithm guarantees that the computation time is still acceptable. Even for large instances, the maximum number of nodes can be evaluated within 5 to 10 minutes on a pentium PC 400 MHz.

6 A Beam Search variant of the algorithm

In this section, we will show how the Branch & Bound algorithm, as discussed in the previous section, can be used to design a Beam Search algorithm. First, we will give a general description of Beam Search, followed by the algorithm we propose for solving the integrated scheduling problem.

6.1 Beam Search

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. In fact, the number of generated nodes is $\mathcal{O}(bw \cdot n^2)$, where n is the number of jobs and bw is

the beam width. Clearly, at every level we generate $\mathcal{O}(bw \cdot n)$ new branches (nodes) and the total tree consists of n levels. 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.

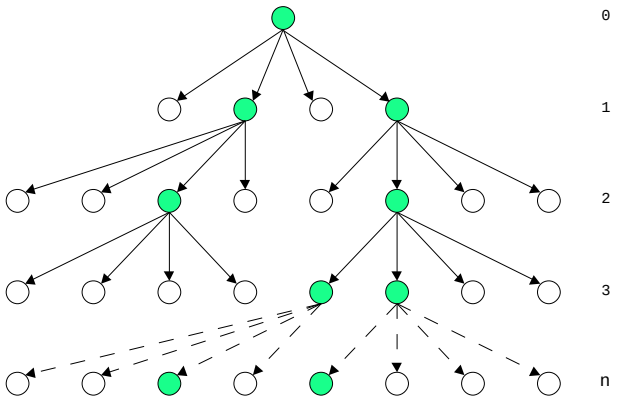

Figure 2: Beam Search algorithm with beam width equal to 2

Beam Search was first used within the field of artificial intelligence. More particular, it was used for speech recognition problems by Lowerre (1976). Within the area of scheduling, Beam Search has been applied by Ow & Morton (1988), Sabuncuoglu & Karabuk (1998), Sabuncuoglu & Bayiz (1999) and Kim & Kim (1999*b*).

One of the main issues when using Beam Search, is the selection of the nodes that will be expanded. Since we only expand a limited number of nodes at each level, this has to be done carefully. However, careful evaluation of a node can be time consuming. Therefore, several authors propose a two-stage 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) left. These nodes are evaluated again, now using a more detailed, time consuming evaluation function.

In the next three subsections, we will develop a Beam Search algorithm for the integrated scheduling problem of handling equipment. This algorithm is based on the Branch & Bound algorithm presented in Section 5. We will discuss the main ingredients of the algorithm, that is, the evaluation of the nodes, the actual selection of nodes and the filtering. The remaining part of the algorithm consists of a straightforward breadth-first-search strategy.

6.2 Evaluation of nodes

The main ingredient of the Beam Search algorithm is the evaluation function which is used to select the nodes that will be further expanded. In our algorithm, we distinguish between nodes on the basis of their lower bound, that is the largest of the three lower bounds as discussed in Subsections 5.3.1 to 5.3.3. Since at level k, all nodes represent an assignment schedule for the first k jobs, the lower bound gives an indication of how "efficient" the first k jobs are scheduled. In case of ties, we consider the upper bound of the nodes, which is calculated for every node.

This upper bound equals the objective value of the heuristic solution we obtain by applying the upper bound heuristic for the remaining unassigned jobs, as discussed in Subsection 5.4. Like in the Branch & Bound algorithm, we keep track of the best solution found so far. Obviously, whenever the lower bound of a node exceeds the objective value of the best solution so far, we can fathom this node. Note that since we calculate a heuristic solution during the evaluation of each node, we actually consider quite some feasible solutions. This is contrary to the usual practice with Beam Search, where only a feasible solution is calculated at the leaves of the tree (so only bw feasible solutions are constructed during the search). As a result of this, it may be possible that although a node is not selected to be further expanded (since its lower bound is too high), it still provides us with a good feasible solution.

6.3 Selection of nodes

The selection of the nodes is done by simply pooling all nodes at each level and select the bw most promising ones. Note that this may lead to the selection of nodes that all have the same parent, which may be undesirable. However, computational tests show that this rarely happens. An alternative way of selecting the nodes, as proposed by several authors, would be to select at the first level bw nodes. Next, for each of these nodes, a single path in the tree is generated by selecting exactly one of its children for expansion in each following level of the tree. Note that these paths are disjunct. Hence, we never have that two selected nodes have the same parent. This latter strategy leads to a "greedy" type of solution, obtained by taking a certain fixed initial job (the node at the first level of the tree), and completing the schedule by selecting at each level the most promising job. However, this means that the nature of the search tree is omitted, since the search tree allows us to select a somewhat less promising job at a certain level, which in the end might give us a very good schedule.

The following remark is in order. Whenever we pool the nodes at each level of the tree, running the algorithm with a larger beam width does not necessarily give a solution that is at least as good as in case of a smaller beam width. This can be explained as follows. Since we select a larger number of nodes, we may choose a node that produces very good offspring in the next level of the tree. Hence, we select all these nodes in the next iteration of the algorithm. In the second next iteration, however, it may happen that these nodes produce poor offspring. So, we may end up with a solution that is worse, compared to a solution found using a smaller beam width. However, several computational tests have shown that pooling the nodes at each level gives better results than the alternative of selecting disjunct paths of nodes.

6.4 Filtering

Our Beam Search algorithm uses a filter to reduce the number of nodes that are evaluated thoroughly at each level. This is done in a straightforward way. In each node of the branching tree, the unassigned jobs are sorted in order of non-increasing tail t_i^{agv}. Now consider a node in the upper part of the tree, that is, a node for which only a small number of jobs have been assigned yet. It is unlikely that scheduling the job with the smallest tail next, will give us a good solution. This is because the job with the smallest tail is a job which is scheduled last on some QC. So, if this job is scheduled next, it will occupy an AGV for quite some time. Hence, the

resulting schedule is likely to have a large makespan. Therefore, we apply the following filtering mechanism. Whenever a node is expanded, we only generate a branch for the *fw* unassigned jobs with the largest tails, where *fw* is the filter width. In this way, we reduce the number of nodes to be evaluated and, as a result, speed up the algorithm. On the other hand, the quality of the solution found may decrease compared to the situation in which we do not apply this filter.

Clearly, setting the beam and filter width requires computational testing and fine tuning. In Section 7, we will give computational results for various sizes of the beam and filter width. Based on these results, we determine good choices for the beam and filter width, based on both the quality of the solutions and the computational effort required.

7 Computational experiments

In this section, we report on the results of computational experiments with both the Branch & Bound algorithm and the Beam Search algorithm. Moreover, we will compare the solutions found by the two algorithms. For testing the algorithms, we used small, medium and large sized real-life instances. The small instances consist of only 8 to 20 jobs. The medium and large size problems contain more than 75 and 160 jobs, respectively. All instances were obtained from a detailed simulation model, which was developed in close cooperation with Europe Combined Terminals, the operator of the automated terminals in Rotterdam (see Meersmans *et al.* (1999)). The simulation model allows for experimenting with different layouts of the container terminal, different number and speed of equipment, etcetera. Hence, for the same set of jobs, we can construct different instances by varying the number of AGV's used, the driving speed of the AGV's and the handling speed of the ASC's. For the QC's, the handling speed was not varied, since this is still a manual operation and thus cannot be carried out faster.

All computations were carried out on a pentium PC 400 MHz with 128 Mb memory, running under WindowsNT.

7.1 Results for the Branch & Bound algorithm

Table 1 shows the results of the Branch & Bound algorithm for problem instances of various size.

The first four columns represent the size of the problem, i.e., the number of containers to be handled and the number of handling equipment, specified per type (QC's, ASC's and AGV's). The fifth and sixth column show the best lower bound and best solution found, respectively. The seventh column shows the gap between the best solution and the lower bound. Recall that whenever the number of evaluated nodes reaches $1 \cdot 10^5$, the Branch & Bound algorithm is terminated. Hence, whenever the gap is larger than zero, this means that the Branch & Bound algorithm was terminated after evaluating $1 \cdot 10^5$ nodes, without finding the optimum. The last column gives the running time of the algorithm.

As we can see from Table 1, most of the small size problems are solved to optimality within a few seconds. The results for the medium and large size problems vary. Mostly, we get good solutions, that is, solutions that are guaranteed to be within 5 percent of the optimum. Keep

problem size				Branch & Bound			
n	QC	ASC	AGV	lower	solution	gap (%)	time (sec.)
8	2	2	2	815	815	0.0	< 1
8	2	2	4	780	780	0.0	< 1
20	3	4	4	1034	1134	9.7	39
20	3	4	6	1034	1034	0.0	< 1
20	3	4	8	1034	1034	0.0	< 1
20	3	4	4	985	1005	2.0	54
20	3	4	6	942	942	0.0	1
20	3	4	8	942	942	0.0	< 1
76	4	12	22	1632	1729	5.9	301
76	4	12	24	1639	1695	3.4	312
76	4	12	26	1639	1676	2.3	342
80	4	12	22	1732	1756	1.4	252
80	4	12	24	1732	1732	0.0	3
83	4	27	8	1760	1813	3.0	216
83	4	27	10	1726	1726	0.0	< 1
83	4	27	16	1998	2130	6.6	184
83	4	27	18	1998	2050	2.6	111
83	4	27	20	1998	1998	0.0	1
85	4	27	8	1790	1820	1.7	111
85	4	27	10	1694	1694	0.0	10
85	4	27	18	1875	2015	7.5	381
85	4	27	20	1875	1924	2.6	393
85	4	27	22	1875	1900	1.3	411
85	4	27	24	1871	1900	1.5	442
167	4	27	8	3432	3561	3.8	478
167	4	27	10	3410	3410	0.0	34
167	4	27	20	3702	3828	3.4	475
167	4	27	22	3702	3738	1.0	387
167	4	27	24	3664	3702	1.0	346
168	4	27	8	3289	3499	6.4	658
168	4	27	10	3234	3234	0.0	16
168	4	27	20	3403	3646	7.1	579
168	4	27	22	3389	3562	5.1	514
168	4	27	24	3404	3418	0.4	389

Table 1: Computational results of the Branch & Bound algorithm

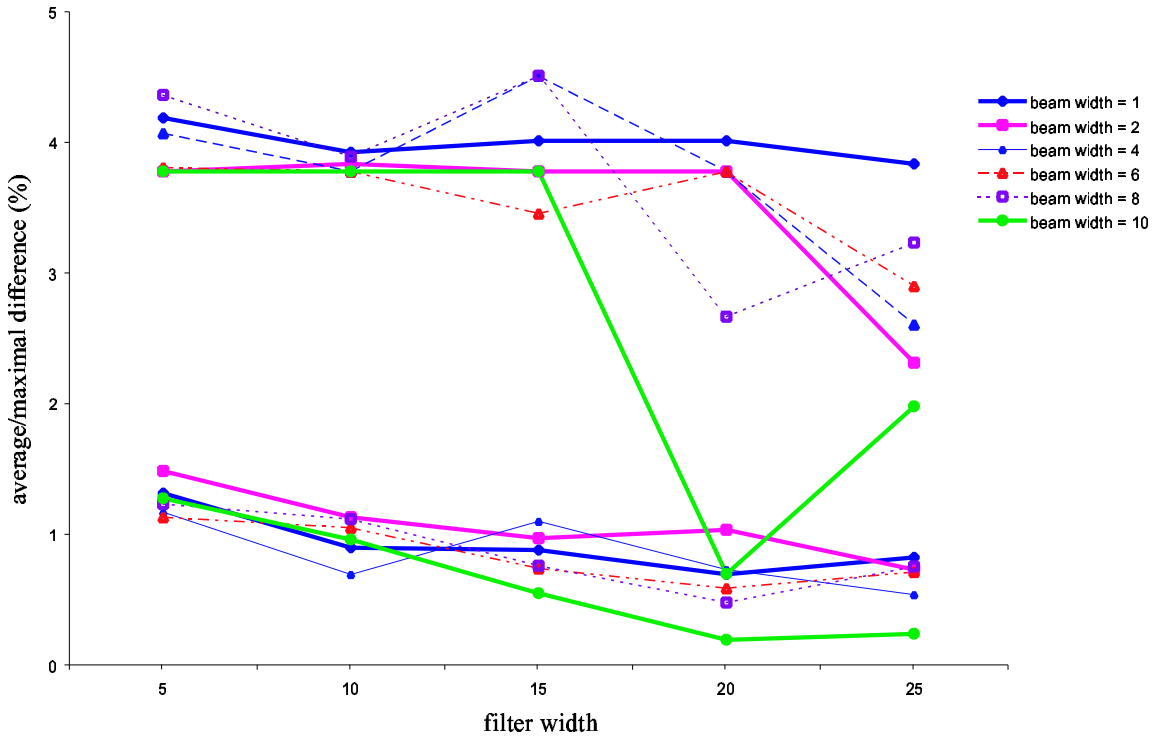

Figure 3: Average and maximal difference between beam and filter width combination and the best beam and filter width combination

in mind that when the gap is larger than 5 percent, the actual deviation from optimality may still be significantly less. Furthermore, we see that the running times also vary, however, for all instances, they are relatively small compared to the makespan they cover (in Table 1, these are also given in seconds), which is important when the algorithm is to be applied in real time. In case the running times are unacceptable, we can resort to the Beam Search algorithm.

7.2 Results of the Beam Search algorithm

In this subsection, we will discuss the results of the Beam Search algorithm. First, we investigate the effect of different beam and filter widths on the quality of the solutions found by the algorithm. Figure 3 shows the results of the Beam Search algorithm for various beam and filter widths[2], based on computational data obtained from solving ten large instances (more than 160 jobs). The figure shows the relative performance of the algorithm with a certain beam and filter width. That is, we compare the solution found by the algorithm with a certain beam and filter width combination, with the best solution found by the algorithm with any choice of these parameters. Note that the best choice of parameter values may vary among the instances.

From Figure 3 we may conclude that the parameter settings of the beam and filter width do

[2]Note that the lines between the points corresponding to different filter widths are only drawn to make it easier to distinguish between the results of the various beam widths. Results were only generated for filter widths that are a multiple of 5

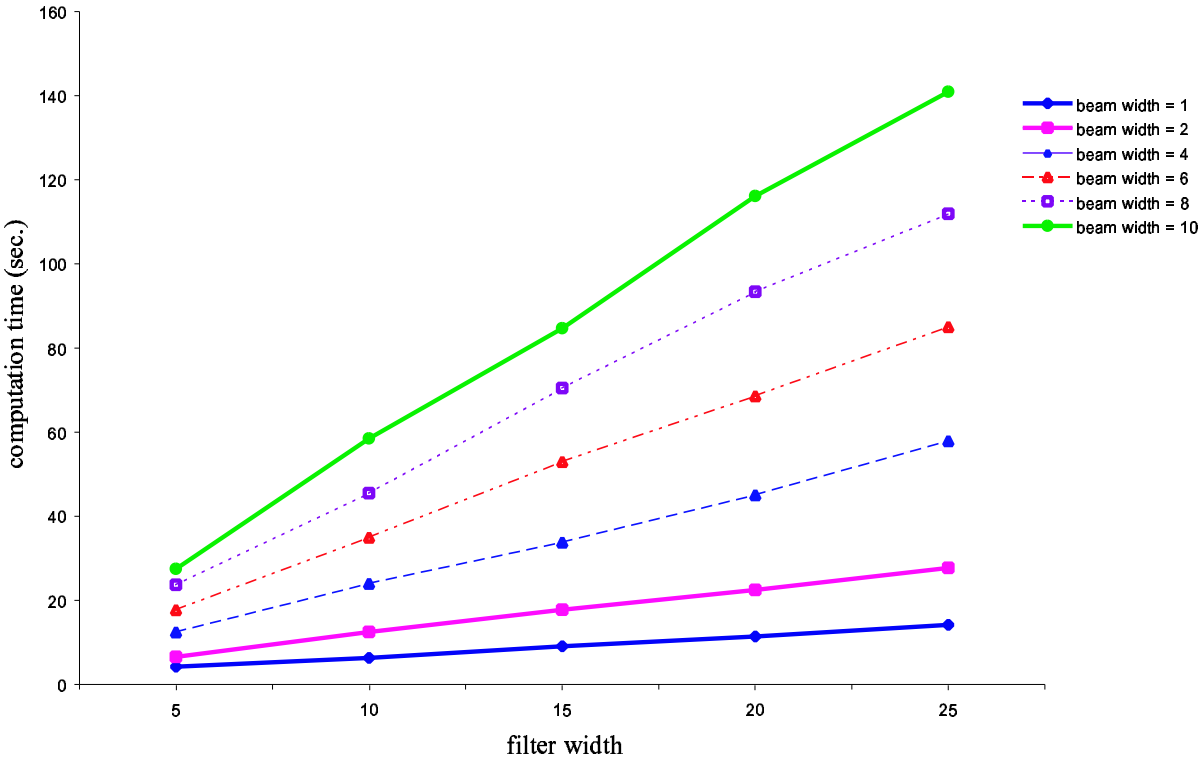

Figure 4: Effects of increasing filter width, for fixed beam width, on computation time

not have a large influence on the results of the algorithm. As we can see, on average, the difference between the results of a given beam and filter width combination and the best beam and filter width combination is within 1.5 percent. As the filter width increases, the differences get even smaller. The figure also shows for each beam and filter width combination its worst case behaviour. So, for each combination, we show the maximum deviation from the best makespan found by any combination, where the maximum is taken over all instances solved. This deviation is for all combinations of the beam and filter width within 5 percent. Hence, we may conclude that the Beam Search algorithm is quite robust and that for practical use, fine tuning of the parameter values for the beam and filter width is not necessary.

Figure 3 also shows that taking a larger beam width does not necessarily lead to a better solution. As pointed out before in Subsection 6.3, this is due to the fact that for a larger beam width, we do not necessarily select the same nodes at each level than for a smaller beam width.

Figure 4 shows the effect on the computation times of increasing the filter width under various beam widths. As we can see from this figure, the computation time grows, for fixed beam width, almost linearly with the filter width. Moreover, doubling the beam width gives approximately a doubling of the computation time. Hence, the number of generated nodes, which is $\mathcal{O}\left(bw \cdot fw \cdot n\right)$ in case of applying a filter, gives a good estimate of the relative computation times under different beam and filter widths.

Combining Figures 3 and 4, it is clear that although the differences in the solution quality of the various beam and filter width combinations may be small, the differences in computation time

time allowed	best Beam Search		
(sec.)	bw	fw	av. deviation (%)
5	1	5	1.3
10	1	15	0.9
30	4	10	0.7
60	4	25	0.5
≥ 120	10	20	0.2

Table 2: Best combinations of beam and filter width for given allowed computation times

are sometimes significant. In Table 2 we show for a given allowed computation time, what the best combination of parameter values for the Beam Search algorithm is. Again, we report the relative performance of the parameter combination with respect to the best combination of the beam and filter width. Note that this best combination of beam and filter width may require more time than allowed.

As we can see from Table 2, for a given time of, for instance, 30 seconds, the combination $bw = 4$, $fw = 10$ yields the best results. For this combination, the average deviation with the best combination (requiring a larger computation time), is within 1 percent. Allowing for more computation time clearly gives better results, however, the improvement is on average only 0.5 percent.

7.3 Comparison of results

In this subsection, we will compare the results of the Branch & Bound algorithm with the results obtained from the Beam Search algorithm. We will set the parameters for the beam and filter width to 4 and 10 respectively. From Table 2 it follows that for a maximal computation time of 30 seconds, this combination gives the best results.

In Table 3, the first four columns specify again the size of the problem. Next, we repeat partially the results of Table 1 for the Branch & Bound algorithm. That is, we give the lower bound, the best solution found and the gap. The last two columns give the results of the Beam Search algorithm, that is, the value of the solution found and the gap with the best known lower bound obtained from the Branch & Bound algorithm.

From Table 3 it follows that the Beam Search algorithm gives solutions that are, in most cases, comparable to the solutions obtained by using the Branch & Bound algorithm, and sometimes even better. Moreover, the Beam Search algorithm requires only about 30 seconds of computation time for the large instances, where the Branch & Bound algorithm usually requires 10 to 20 times as much computation time (see also Table 1).

problem size				Branch & Bound			Beam Search (4, 10)	
n	QC	ASC	AGV	lower	solution	gap (%)	solution	gap (%)
76	4	12	22	1632	1729	5.9	1737	6.4
76	4	12	24	1639	1695	3.4	1688	3.0
76	4	12	26	1639	1676	2.3	1676	2.3
80	4	12	22	1732	1756	1.4	1784	3.0
80	4	12	24	1732	1732	0.0	1762	1.7
83	4	27	8	1760	1813	3.0	1790	1.7
83	4	27	10	1726	1726	0.0	1726	0.0
83	4	27	16	1998	2130	6.6	2120	6.1
83	4	27	18	1998	2050	2.6	2050	2.6
83	4	27	20	1998	1998	0.0	1998	0.0
85	4	27	8	1790	1820	1.7	1832	2.3
85	4	27	10	1694	1694	0.0	1694	0.0
85	4	27	18	1875	2015	7.5	2029	8.2
85	4	27	20	1875	1924	2.6	1924	2.6
85	4	27	22	1875	1900	1.3	1904	1.5
85	4	27	24	1871	1900	1.5	1900	1.5
167	4	27	8	3432	3561	3.8	3534	3.0
167	4	27	10	3410	3410	0.0	3410	0.0
167	4	27	20	3702	3828	3.4	3761	1.6
167	4	27	22	3702	3738	1.0	3746	1.2
167	4	27	24	3664	3702	1.0	3702	1.0
168	4	27	8	3289	3499	6.4	3394	3.2
168	4	27	10	3234	3234	0.0	3234	0.0
168	4	27	20	3403	3646	7.1	3589	5.5
168	4	27	22	3389	3562	5.1	3544	4.6
168	4	27	24	3404	3418	0.4	3415	0.3

Table 3: Computational results of Beam Search algorithm with $bw = 4$ and $fw = 10$

8 Conclusions and further research

In this paper we have considered the integrated scheduling of various types of handling equipment at an automated container terminal. Efficient scheduling of the equipment both reduces the time vessels spend in the port and increases the productivity of the terminal. Therefore, great costs savings are at stake.

We have considered a class of layouts in which all AGV's pass a common point after they are unloaded at the QC's. This is, for instance, the case at the automated container terminals in Rotterdam. For such layouts, we have shown that we may restrict ourselves to schedules in which the assignment order of the jobs on the AGV's is consistent with the order of the jobs on each of the ASC's. Hence, the assignment order of the jobs on the AGV's determines the schedule completely. This has led to the development of a Branch & Bound algorithm. Within the algorithm, a number of combinatorial lower bounds is used to cut off unpromising nodes in the search tree. These lower bounds are easy to calculate. Nevertheless, they turn out to be very effective since the algorithm not only performs well on small instances, but also on moderate and large real-life problems, as the results of Section 7 show. The running times are acceptable, especially when compared to the length of the period for which the schedule is made. This is important when rescheduling in real time is necessary, because of deviations from the expected processing times (see also below).

Based on the Branch & Bound algorithm, we also developed a Beam Search algorithm. This Beam Search algorithm gives similar, or even slightly better, results as the Branch & Bound algorithm. However, especially for large problem instances, the required computation time of the Beam Search algorithm is far less. Moreover, the Beam Search algorithm turns out to be robust with respect to the choice of the beam and filter width. Hence, the algorithm appears to be practically applicable without extensive fine tuning.

Future research will focus on applying the algorithms presented in this paper within a dynamic context. So far, we have assumed that all processing times are deterministic. In practice however, these processing times can vary, especially the processing times of the QC's, since this is still a manual operation. Also the driving times of the AGV's may vary since there may be interaction between vehicles. For instance, if two AGV's want to use the same track, one has to wait until the other has passed. Because of these deviations from the planned processing times, rescheduling might be advantageous. On the other hand, rescheduling too often may also not give satisfactory results. It is therefore interesting to investigate the effects of the frequency of rescheduling on the overall performance. Furthermore, extension of the model by introducing additional tasks for the ASC's, like reorganisation moves, land side moves, etcetera, will be a topic of further research.

References

Ascheuer, N., Fischetti, M. & Grötschel, M. (2000*a*). *A polyhedral study of the asymmetric travelling salesman problem with time windows.* Networks, 36:69–79.

Ascheuer, N., Grötschel, M. & Abdel-Aziz Abdel-Hamid, A. (1999). *Order picking in*

an automatic warehouse: solving online asymmetric tsps. Mathematical Methods of Operations Research, 49:501–515.

Ascheuer, N., Jünger, M. & Reinelt, G. (2000*b*). *A branch & cut algorithm for the asymmetric hamiltonian path problem with precedence constraints.* Computational Optimization and Applications, 17:61–84.

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.

Bilge, U. & Ulusoy, G. (1995). *A time window approach to simultaneous scheduling of machines and material handling system in an fms.* Operations Research, 43:1058–1070.

Chen, Y., Leong, T-Y., Ng, J.W.C., Demir, E.K., Nelson, B.L. & Simchi-Levi, D. (1997). *Dispatching automated guided vehicles in a mega container terminal.* The National University of Singapore/Dept. of IE & MS, Northwestern University.

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.

Evers, J.J.M. & Koppers, S.A.J. (1996). *Automated guided vehicle traffic control at a container terminal.* Transportation Research Part A, 30:21–34.

Graham, R.L., Lawler, E.L., Lenstra, J.K. & Rinnooy Kan, A.H.G. (1979). *Optimization and approximation in deterministic sequencing and scheduling: a survey.* Annals of Discrete Mathematics, 4:287–326.

Johnson, M.E. & Brandeau, M.L. (1993). *An analytic model for design of a multivehicle automated guided vehicle system.* Management Science, 39:1477–1489.

Kim, K.H. & Bae, J.W. (1999). *A dispatching method for automated guided vehicles to minimize delays of containership operations.* International Journal of Management Science, 5:1–25.

Kim, K.H. & Kim, K.Y. (1999*a*). *An optimal routing algorithm for a transfer crane in port container terminals.* Transportation Science, 33:17–33.

— (1999*b*). *Routing straddle carriers for the loading operation of containers using a beam search algorithm.* Computers and Industrial Engineering, 36:109–136.

Lowerre, B.T. (1976). *The HARPY speech recognition system.* Ph.D. thesis, Carnegie-Mellon University.

Meersmans, P.J.M., van Hoesel, C.P.M. & Wagelmans, A.P.M. (forthcomming). *Integrated scheduling of handling equipment at container terminals.* Working paper.

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

OW, P.S. & MORTON, T.E. (1988). *Filtered beam search in scheduling.* International Journal of Production Research, 26:35–62.

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

SABUNCUOGLU, I. & BAYIZ, M. (1999). *Job shop scheduling with beam search.* European Journal of Operational Research, 118:390–412.

SABUNCUOGLU, I. & KARABUK, S. (1998). *A beam search-based algorithm and evaluation of scheduling approaches for flexible manufacturing systems.* IIE Transactions, 30:179–191.

THONEMANN, U.W. & BRANDEAU, M.L. (1997). *Designing a zoned automated guided vehicle system with multiple vehicles and multiple load capacity.* Operations Research, 45:857–873.

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

VIS, I.F.A., KOSTER, R. DE, ROODBERGEN, K.J. & PEETERS, L.W.P. (2001). *Determination of the number of automated guided vehicles required at a semi-automated container terminal.* Journal of the Operational Research Society, 52:409–417.

WINTER, T. (1999). *Online and real-time dispatching problems.* Ph.D. thesis, Technischen Universität Braunschweig.

YU, W. (1996). *The two-machine flow shop problem with delays and the one-machine total tardiness problem.* Ph.D. thesis, Eindhoven University of Technology.

Appendix A Proof of Theorem 4.1

Proof: The problem of minimizing the makepan on a single machine, subject to chain-like time-lags is known to be NP-hard, even if all processing times are equal (Yu (1996)). Following the notation of Graham *et al.* (1979) we can denote this scheduling problem by $1|chains(l_{ij}); p_i = 1|C_{max}$, where $l_{i,j}$ denotes the minimum time-lag between the completion of job i and its immediate successor job j. In the recognition version of this problem, the question is whether or not there exists a feasible schedule such that the makespan does not exceed a given value $\bar{C}_{max}$.

Given an instance of the above scheduling problem, we construct the following instance of our integrated scheduling problem.

1. For every job in the instance of $1|chains(l_{ij}; p_i = 1)|C_{max}$, we have a container.

2. All these containers belong to the same stack and their ASC processing times are equal to $\epsilon < 1$.

3. There is a one-to-one correspondence between chains and QC's. The order in which a QC handles its containers is exactly the order defined by the corresponding chain. If container i is the immediate predecessor of container j in one of these chains, then the processing time of i on the QC is equal to $l_{ij} + 1$.

4. There is one AGV. The AGV is initially located at the common point, which is also the point where the QC's lift the containers from the AGV. To drive from this common point to the stack takes more than ϵ time. The total time to drive from the common point to the stack, pick up a container, drive back to the common point and lift the container from the AGV is equal to 1.

5. The question is whether there exists a feasible schedule such that the makespan does not exceed $\bar{C}_{max}$.

Note that the processing times on the ASC are so small that they do not affect the scheduling.

A feasible schedule for the $1|chains(l_{ij}; p_i = 1)|C_{max}$ instance, can easily be translated into a feasible schedule of the integrated scheduling problem that has the same makespan, by letting the AGV leave from the common point at the starting time of the corresponding job. Note that if i and j are two containers that need to be processed consecutively on the same QC, then the AGV will not leave the common point to pick up container j before l_{ij} time units have passed since it has dropped off container i at the QC.

Now suppose that we have a feasible schedule for our integrated scheduling problem, then there exists a feasible schedule with the same makespan in which all the waiting time of the AGV occurs just before it leaves the common point to pick up a container. (If waiting time occurs at another point, then it can be eliminated by letting the AGV leave later.) It is now easy to see that by using the same correspondence as before, we obtain a feasible schedule for the $1|chains(l_{ij}; p_i = 1)|C_{max}$ instance with the same makespan.

We have now shown that the integrated scheduling problem is NP-hard if there is a single AGV. We will now sketch how one can prove this result also for multiple AGV's. Add to the constructed instance above an arbitrary number of AGV's and the same number of containers. All these containers are located in a second stack and all need to be transported to an additional QC that is located "far away" from the second stack and from the other QC's. The processing times of the additional containers on the ASC and QC are chosen very small. The idea is to choose the transportation time from the ASC to the QC such that the makespan is exactly equal $\bar{C}_{max}$ if all additional AGV's leave almost immediately to each pick up one of the additional containers. We leave it to the reader to work out the details.

□

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

2000

A Greedy Heuristic for a Three-Level Multi-Period Single-Sourcing Problem
H. Edwin Romeijn & Dolores Romero Morales
ERS-2000-04-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 Decision Making in Marketing Management
F&A Financial Decision Making and Accounting
STR Strategic Renewal and the Dynamics of Firms, Networks and Industries

Integer Constraints for Train Series Connections
Rob A. Zuidwijk & Leo G. Kroon
ERS-2000-05-LIS

Competitive Exception Learning Using Fuzzy Frequency Distribution
W-M. van den Bergh & J. van den Berg
ERS-2000-06-LIS

Models and Algorithms for Integration of Vehicle and Crew Scheduling
Richard Freling, Dennis Huisman & Albert P.M. Wagelmans
ERS-2000-14-LIS

Managing Knowledge in a Distributed Decision Making Context: The Way Forward for Decision Support Systems
Sajda Qureshi & Vlatka Hlupic
ERS-2000-16-LIS

Adaptiveness in Virtual Teams: Organisational Challenges and Research Direction
Sajda Qureshi & Doug Vogel
ERS-2000-20-LIS

Assessment of Sustainable Development: a Novel Approach using Fuzzy Set Theory
A.M.G. Cornelissen, J. van den Berg, W.J. Koops, M. Grossman & H.M.J. Udo
ERS-2000-23-LIS

Applying an Integrated Approach to Vehicle and Crew Scheduling in Practice
Richard Freling, Dennis Huisman & Albert P.M. Wagelmans
ERS-2000-31-LIS

An NPV and AC analysis of a stochastic inventory system with joint manufacturing and remanufacturing
Erwin van der Laan
ERS-2000-38-LIS

Generalizing Refinement Operators to Learn Prenex Conjunctive Normal Forms
Shan-Hwei Nienhuys-Cheng, Wim Van Laer, Jan Ramon & Luc De Raedt
ERS-2000-39-LIS

Classification and Target Group Selection bases upon Frequent Patterns
Wim Pijls & Rob Potharst
ERS-2000-40-LIS

Average Costs versus Net Present Value: a Comparison for Multi-Source Inventory Models
Erwin van der Laan & Ruud Teunter
ERS-2000-47-LIS

Fuzzy Modeling of Client Preference in Data-Rich Marketing Environments
Magne Setnes & Uzay Kaymak
ERS-2000-49-LIS

Extended Fuzzy Clustering Algorithms
Uzay Kaymak & Magne Setnes
ERS-2000-51-LIS

Mining frequent itemsets in memory-resident databases
Wim Pijls & Jan C. Bioch
ERS-2000-53-LIS

Crew Scheduling for Netherlands Railways. "Destination: Curstomer"
Leo Kroon & Matteo Fischetti
ERS-2000-56-LIS