Optimization
of
Container Handling Systems

ISBN: 90-5170-591-3
Cover design: Crasborn Graphic Designers bno, Valkenburg a.d. Geul

The book is no. 271 of the Tinbergen Institute Research Series. This series is established through cooperation between Thela Thesis and the Tinbergen Institute. A list of books which already appeared in the series can be found in the back.

Optimization
of
Container Handling Systems

Optimalisatie van containeroverslagsystemen

Proefschrift

ter verkrijging van de graad van doctor
aan de Erasmus Universiteit Rotterdam
op gezag van de Rector Magnificus
Prof.dr.ir. J.H. van Bemmel
en volgens besluit van het College voor Promoties

De openbare verdediging zal plaatsvinden op
vrijdag 12 april 2002 om 16.00 uur
door

Patrick Johannes Maria Meersmans

geboren te Elsloo

Promotiecommissie

Promotor:	Prof. dr. ir. R. Dekker
Overige leden:	Prof. dr. ir. M.B.M. de Koster
	Prof. dr. S.L. van de Velde
	Prof. dr. ir. C.P.M. van Hoesel
Co-promotor:	Prof. dr. A.P.M. Wagelmans

Acknowledgements

On a windy day in september 1996, I visited the Econometric Institute for the first time. It was Albert Wagelmans who I first met and talked to about doing a PhD in Rotterdam. Throughout the years, Albert has been great sparring partner for ideas. Always asking the right questions at the right time and making sure that I worked out the details in a proper way. I am indebted to him a lot and I want to thank him for all the support he gave me during my PhD project.

It was Rommert Dekker, my promoter, who gave me the opportunity to do my PhD and initiated my research project. Throughout the years, I learnt a lot from his rich practical experience and I am very grateful for that.

Although not involved officially, Stan van Hoesel kept interested in my PhD project. Stan was always prepared on a friday afternoon to discuss my research and to show me some directions. His everlasting optimism really got me through some tough times! I want to thank him and the other members of the doctoral committee, Steef van de Velde and Rene de Koster, for reading my thesis and making valuable comments.

My colleagues at the Econometric Institute made me feel at home immediately. Especially, I would like to thank Marcel for playing guitar and being a great roommate, Raymond for endless stories, Gonda for coffee and IFORS99, Arjan for black-and-white remarks, Emöke for Pálinka, Ovidiu for discussions, Peter for darts, Jos for three o'clock tea sessions, Dennis for CPLEX, Philip for a great Danish beach holiday, and Marisa, Arianna, Gabriella and Csaba for adding an international touch.

Also thanks to my friends from the University of Maastricht, especially Jan-Willem for programming tips, mosquito repellent and "TABU" search. And of course Dolores, for dinner parties, exclamation marks, kroketten, and showing me around in the beautiful city of Sevilla.

Finally, I want to thank Leon Peeters. After studying together in Maastricht and doing our Masters thesis at the same company, we both ended up in Rotterdam to do a PhD. So, who knows where we will meet again? I want to thank him for all his help on LaTeX and LEDA problems, being a travel mate to several conferences, and most of all, being a friend.

En noe aan 't ènj gekóme, wil ich de luuj bedanke die aan de weeg höbbe gesjtaon van dit proofsjrif. Veur al hun sjteun en hulp. Pap en mam, bedank!

Patrick Meersmans
Rotterdam, April 2002

Contents

Chapter 1

Introduction

The skyline of every major port is nowadays dominated by stacks of colorful steel boxes: containers. The invention of the container is attributed to Malcolm McLean, who started the movement of trailers on ships (with the Sea/Land line) and rail from New York to the Gulf Coast in 1956 (see Chadwin *et al.* (1990)). Later on, the trailer changed to containers. The container has increased port productivity enormously and changed the appearance of harbors from large areas with breakbulk cargo to long stretches of bulkheaded waterfront with many large cranes serving big container ships. The container has also allowed wages to be increased considerably, and with the container ICT has made its inroad into harbors.

The introduction of the container not only led to a revolution in transportation, the whole world trade was affected. Due to its homogeneity, competition in container transport is only possible on price. This resulted in low margins, which were answered by increasing the scale of the operations. As a result, the container has made transport cheap. Due to these low transportation costs, every country in the world is now able to take part in the global production of goods. Especially the Asian economies have benefited from this. It is no coincidence that the rise of the "Asian Tigers" has taken equal steps with the development of containerization. This is illustrated in Figure 1.1, which shows the number of containers handled in the port of Singapore and the development of the Gross National Product (GNP). From the figure we can see that the development of the GNP keeps equal pace with the volumes of containers handled.

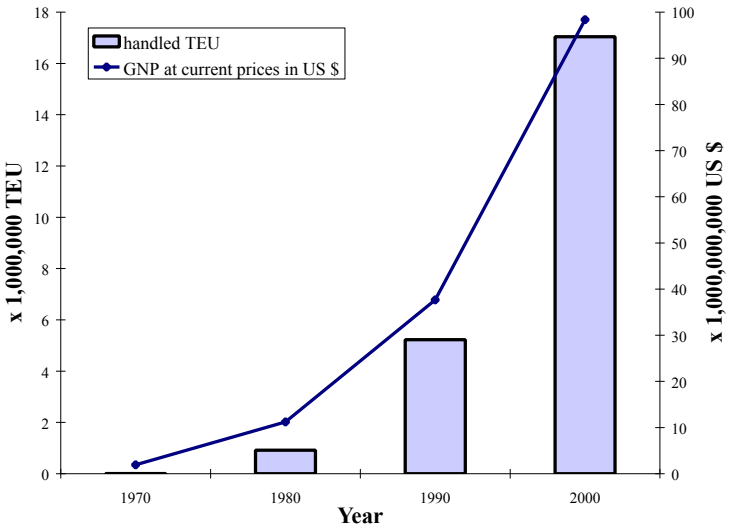

Figure 1.1: Development of GNP and container volumes handled, Singapore, 1970 - 2000 (source: Containerisation International Online; Statistics Singapore)

1.1 Containers

Containers are steel boxes of a standard size. The standard width of a container is 8 ft. and its height is 8.6 ft. There are three different lengths, 20, 40 and 45 ft. Volumes of containers are usually expressed in TEU, which stands for Twenty foot Equivalent Unit, i.e., a container of 20 ft. Today, there are about 6 million containers, enough to put a double wall around the globe.

Almost every product can be transported in containers, from traditional break-bulk like cotton, to high value electronics. There exist also special containers for refrigerated goods, bulk, chemicals, etcetera.

1.2 Facts and figures

Ever since its introduction in the 1960s, the number of containers handled worldwide has grown almost every year with double digits. Figure 1.2 shows the increase in the number of containers handled in the 10 major container ports over the last 30 years. It was estimated that in total, around 168 million TEU were handled worldwide in the year 2000 (see Containerisation International Online).

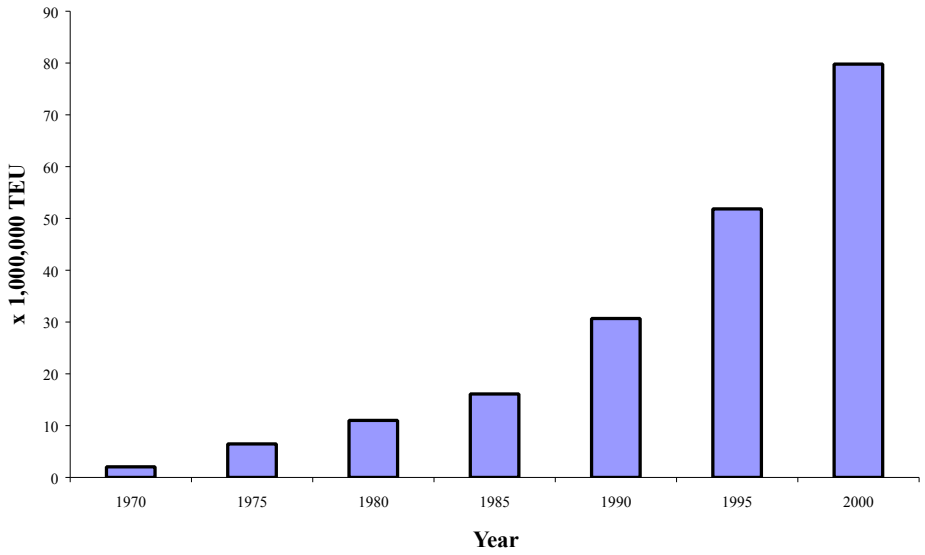

Figure 1.2: Volumes handled in the 10 major container ports (source: Containerisation International Online)

Together with the enormous increase in the number of containers handled, also the vessel size has increased enormously. The first container vessels were converted cargo carriers from World-War II. These vessels were only able to carry a few tens of containers. Moreover, they usually had crane facilities on board, since terminal facilities were not available everywhere. The next generation of container vessels had a capacity ranging from a few hundred TEU to a few thousand TEU. These so-called Panamax vessels are still able to pass the Panama channel. A turning point was the introduction of the so-called Post Panamax vessels in the 1980s. These vessels are so big that they are not able to pass the Panama channel. Such a Post Panamax vessel is able to carry more than 4,000 TEU.

The next breakthrough in ship size was established in 1997 with the "Sovereign Maersk", the first of a series of vessels able to carry more 6,000 TEU. And in the near future, vessels up to 10,000 TEU are expected (see e.g. Vickerman (1998)). In fact, it is said that with current technology, a vessel of 15,000 TEU should be possible. However, such vessels are not able to pass the Suez channel, which will lead to significant longer travel times between Asia and Europe. It is therefore questionable whether such a vessel will ever be built, although there was a similar scepticism with the introduction of the Post Panamax vessels.

The operational costs of the current generation of container vessels are high (over $ 1,000 per hour). As larger ships are expected, these costs will increase. This will put increasing pressure on terminals to speed up their handling, since the economies of scale of larger ships can only be realized if the handling speed at the harbors increases accordingly (see Cullinane and Khanna (2000)).

1.3 Trends in container handling

In the last 50 years, the container shipping business has made a stormy development. Growth rates in double figures appeared almost every year. For the coming years, the expected growth rates are somewhat smaller, but still around 5 percent per year.

The trend of increasing ship size is likely to continue, despite a significant overcapacity in the market, which is as old as the container itself. Together with larger ships, there is also increasing cooperation between the various shipping lines. This resulted in alliances. The next step will be the merger of shipping lines. This process is already going on with the mergers of P&O and Nedlloyd and the takeover of Sealand by Maersk, resulting in the two largest container shipping lines today. Experts expect that this concentration will finally result in around five major players.

Together with the introduction of larger vessels, also the number of ports at which these vessels call is likely to decrease. As vessels increase in size, their operating costs are rising. In order to achieve the economies of scale of such vessels, their in-port time should decrease. This can be achieved by calling fewer ports, or to increase the productivity of loading and unloading operations (see also Cullinane and Khanna (2000)). Hence, there will be a trend to so-called hub-and-spoke networks, in which there is a point to point connection between two major ports on different continents (for instance Rotterdam - Hong Kong). The containers destined for other ports are then transported using so-called short sea vessels. Consequently, such a mega hub becomes more and more a transhipment terminal of flows destined for other terminals, covering a large hinterland (continent) in this way. A good example of this is the port of Singapore, at which more than 80 percent of the incoming containers leaves the terminal again via water, destined for another port in Asia.

As vessels and their call size increase, container terminals have to grow accordingly, providing sufficient resources for fast loading and unloading of the vessels. Moreover, as already mentioned, the handling of such large vessels should be faster, in order to preserve the economies of scale of such large vessels. Obviously, one can increase the number of handling equipment, although there is a limit to the number of cranes that can be deployed to handle a ship. Hence, there should be technical improvements such that the handling speed of cranes is increased. Since there is

also a physical limit to this, a more efficient use of the equipment will also lead to increased productivity.

Next to the ongoing development of faster handling equipment, the new trend is automation of the handling equipment. A pioneering role in this is played by the Rotterdam based operator Europe Combined Terminals (ECT), which established the first automated container terminal in 1993, dedicated to handle vessels of the Sealand shipping company. At the moment, a second automated terminal has been established by ECT and within the coming years, a few other of these terminals will be built. Also the port of Hamburg is currently building an automated container terminal, almost similar to the ECT terminals. This terminal is expected to be operational by the year 2003. Moreover, also the ports of Singapore and Antwerp are making plans for automation of the handling equipment.

1.4 Aim and outline of the thesis

The aim of this thesis is to provide models and algorithms for efficient scheduling of terminal handling equipment. In particular, we will consider automated handling systems. Efficient scheduling of handling equipment is crucial, especially in automated systems. First of all, since there is no driver, there has to be a formal scheduling method, "instructing" the automated equipment exactly which tasks to perform and in which order. Moreover, since drivers experience is missing, efficient scheduling is crucial to achieve satisfactory overall performance of the terminal. Furthermore, better scheduling results in smaller investments in expensive equipment.

The scheduling of terminal equipment is typically an operational problem. It is necessary for the operation of the terminal to have a detailed schedule for each piece of equipment that is operating on the terminal. Although the quality of the scheduling certainly has an influence on more tactical or even strategic decisions (e.g., the investment in handling equipment), we will not deal with these issues in this thesis explicitly.

As far as we know, there is no literature available which deals with integrated scheduling of various types of handling equipment at a container terminal. Obviously, scheduling the equipment in an integrated way can lead to much better performance than in case the scheduling is not coordinated. Besides, uncoordinated scheduling of handling equipment may even lead to overall infeasibility, i.e., deadlock situations. Our goal is therefore to provide algorithms that produce feasible schedules under any circumstance (except in case of breakdowns of handling equipment).

The thesis is organized as follows. In the next chapter, we will give an extensive overview of the problems in container handling that were already dealt with by the

Operations Research community. In this literature overview, we will consider all the processes that take place at the terminal: from the berthing of vessels, stacking of containers, scheduling of equipment, to the interface of the terminal with the landside.

In Chapter 3, we will give a detailed description of the scheduling problem of handling equipment at automated container terminals. We will describe the characteristics of the equipment and discuss the scheduling problem that appears during the loading operation of a single vessel. For the specific problem setting which reflects the current generation of automated container terminals, we develop an exact Branch & Bound algorithm for solving the integrated scheduling problem. This is done in Chapter 4, in which we also discuss a Beam Search heuristic that is based on the Branch & Bound algorithm. Computational experiments are done with both algorithms, in which we assume a static environment, i.e., we assume that all information about the containers to be handled is known in advance. In Chapter 5, we consider a dynamic case, i.e., we assume that information about handling times of containers is uncertain and moreover, new containers may appear within the planning horizon. For this dynamic case, we test the performance of the Beam Search algorithm of Chapter 4. Moreover, we develop a number of dispatching rules. The performance of these dispatching rules is compared with the performance of the Beam Search algorithm.

A model for more general layouts of container terminals is presented in Chapter 6. For this model, we develop a Mixed Integer Programming (MIP) formulation. Moreover, for this MIP formulation, we derive several classes of valid inequalities. We test the effectiveness of these valid inequalities, i.e., the reduction of the number of nodes in the search tree. Finally, in Chapter 7, we will summarize the thesis, make concluding remarks and give some directions for further research.

Chapter 2

An introduction to container logistics

Although container logistics has been a niche area in science, its societal importance has given rise to several research projects in Europe, the USA and Asia, and as a result, it is gaining more and more attention in the scientific area. In this chapter, we will give an overview of the use of operations research models and methods in the design and operation of container terminals. We will describe the activities that take place at a container terminal and give an overview of the relevant decision problems at a strategic, tactical and operational level. For each of these problems the appropriate operations research contributions are discussed.

We base the overview on three sources, viz. publicly available papers in scientific journals, papers in professional publications, Master and PhD theses made available and our own experiences with projects on container handling within the port of Rotterdam. We refer to Chadwin *et al.* (1990) for a basic introduction to container transportation. We do not tackle every aspect of container logistics, like the positioning of empty containers over the continents. Nor do we discuss terminal issues for which no quantitative models have been published, like multi- versus single user terminals and location selection. Very few overviews are available to date. A first attempt was made by Vis and De Koster (1999).

The structure of this chapter is based on following the flow of sea containers through a terminal. After a general introduction in Section 2.1, we start by discussing how containers are stowed in the vessel (stowage planning) in Section 2.2. Next, in Section 2.3, we discuss the berth allocation problem (the assignment of container vessels to berths) and the allocation of quay cranes for unloading and loading of

the vessels. We continue in Section 2.4 with a discussion of the literature related to the scheduling of handling equipment during the (un)loading operation of a vessel. After that, in Section 2.5, we discuss the stacking, that is, the temporary storage of containers at the terminal. Finally, we discuss the landside interface and some overall terminal studies in Section 2.6, followed by some conclusions in Section 2.7.

2.1 Container handling at terminals - description and problem overview

In this section, we will roughly discuss the activities that take place at a container terminal. Moreover, we will discuss shortly the kind of decision problems that appear at each level of the decision hierarchy (strategic, tactical, operational).

2.1.1 Description of activities

In this subsection, we give a short overview of activities at a container terminal. In this we follow the sequence of events from the arrival of a ship until the departure of a container to an inland destination. We will briefly introduce the material handling equipment. A container terminal is based on a maritime interface. Hence, it has quays with cranes to load and unload ships, a stack to store containers and an interface with inland transportation either by truck, rail or barge. The quay cranes are very large in order to handle the containers from sea-going vessels. The size of the latest generation of vessels is enormous: typically more than 300 m long and 42 m wide, such that 17 boxes can be stored across on deck (P&O Nedlloyd N series). Often, there are smaller cranes to serve the feeder ships or the inland barges. Most quay cranes consist of an open structure with a beam extending over the ship. They have a trolley and a spreader to attach to the containers from the top, which are then moved by cables.

Once from the ship, the containers are put on a vehicle, e.g. a trailer or an automated guided vehicle (AGV), which moves the container to the marine stack. This stack is the main decoupling point between the import and export flows, either from sea to sea or from sea to land and vice-versa. The stack may consist of containers on chassis (quite often in the USA), or, of containers stacked on top of each other in a certain pattern (more common in Europe and Asia, because of space restrictions). Normally only containers of the same type are put on each other. The stacking itself occurs by stacking cranes or straddle carriers. Stacking cranes can be rail-mounted (which is stable, fast, but unflexible), rubber-tired (more flexible) or be put on a concrete or steel structure (overhead bridge cranes). Besides manually operated

stacking cranes, there are also (semi-)automated stacking cranes. An alternative system for the transportation of containers to the stack is the use of straddle carriers. This equipment combines the functionalities of a stacking crane and a transport vehicle. A straddle carrier is able to drive over a container, lift it up (3 or 4 containers high) and move it around. Figure 2.1 shows some terminal equipment.

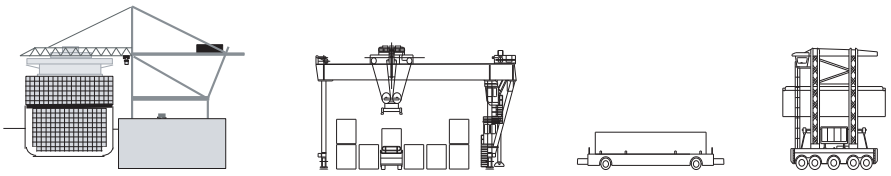

Figure 2.1: Terminal equipment: quay crane, stacking crane, AGV, straddle carrier

The stack on the ground is usually organized into a block pattern that is served by one or more cranes. One main problem with stacking containers on top of each other is that when a bottom container is needed, the containers above it need to be put at another position in the stack (called reshuffling). A random-access storage system, like an Automated Storage/Retrieval System in a distribution warehouse would circumvent these problems. Yet to date none operate for maritime containers. Several (unpublished) studies were carried out on these systems but the problem is that containers are much heavier (up to 30 metric tonnes) and larger compared to a pallet (one ton), which makes the construction of such a storage system prohibitively expensive.

A container terminal can have several interfaces with inland transportation. First, there are transfer points for trucks, which are then loaded from the stack using straddle carriers, reach stackers or other cranes. Next, there can be a rail terminal or service center, where containers are loaded onto or from trains. Finally, there can be a barge service center where barges are loaded using specific equipment.

In Figure 2.2, a schematic view of a typical terminal is given, with all its main components like the quays, the stack and the landside interface.

2.1.2 Classification of decisions

Decisions on container handling equipment and its scheduling can be categorized according to the time horizon involved. Usually, one distinguishes between strategic, tactical and operational decisions. Specific at the strategic level is the design of the

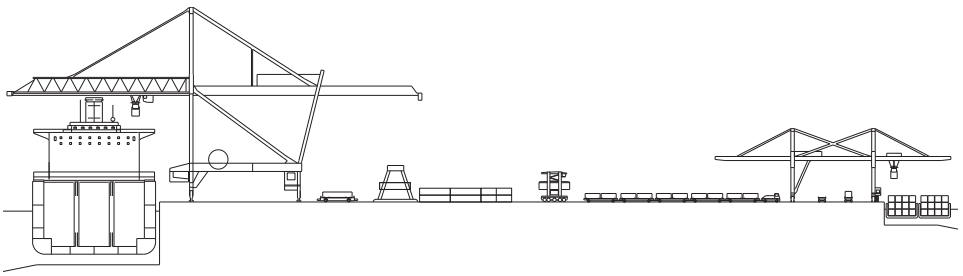

Figure 2.2: Schematic view of a container terminal

terminal. There, one makes decisions on the choice of the handling equipment, on the way (or concept) in which operations are carried out as well as on the layout of the terminal. At the tactical level (months/weeks), one decides on capacity levels of equipment and manning, on layout structures and on timetables of ships and trains (in terms of frequency and day of week). Finally, at the operational level, one decides on the allocation of capacity to the work to be done. One may even add a real-time level, where quite detailed control decisions have to be made regarding automated equipment, like which route to take for an AGV, which speed to use and when to make turns. We will only sideways deal with these issues in this chapter.

2.2 Container stowage

Container ship stowage planning deals with the arrangement of containers inside the ship. In general, a container ship calls at a number of ports on its route and in each port, containers are unloaded and loaded. The containers that are loaded are destined for subsequent ports along the route of the vessel. The stowage problem handles the assignment of containers to positions in the ship. This has to be done is such a way the first of all, the overall stability of the ship is maintained and secondly, the number of unnecessary movements is minimized. These unfavorable movements appear if at a certain port, containers have to be unloaded and reloaded again, since they are stored on top of containers destined for this port.

Containers differ in weight and size (20, 40 and 45 ft.). The placement of the containers in the ship should be such that the stability, torsion and strength of the vessel is maintained. Obviously, heavy containers are usually stored at the bottom of the vessel. The light containers are then stacked upon them. Moreover, containers of different size cannot be stacked on top of each other. Next to the weight and size considerations, a number of other restrictions has to be taken into account. For

instance, reefer containers require electrical power, which is only available at specific positions in the ship. Certain containers carrying dangerous materials (with a certain IMO classification) have specific stowage requirements, which include separate stowage from other containers. Next to these technical restrictions, also the destination of containers has to be taken into account when the stowage plan is made. Obviously, if a vessel calls port A first and then port B, it is not desirable to stack containers destined for port B on top of containers for port A. This will lead to so–called shifting moves.

Avriel *et al.* (1998, 2000) focus on the minimization of the number of shifting moves. The technical constraints on the stability of the vessel are left out in the model. In Avriel *et al.* (1998) a 0-1 programming formulation is given where the objective is to minimize the number of shifting moves. However, the problem is NP-hard (see also Avriel *et al.* (2000)) and solving the model to optimality can not be done in a reasonable amount of time for real-life problems. As a result, a heuristic procedure is proposed. In a companion paper, Avriel *et al.* (2000) elaborate on the relation between stowage planning and the coloring of circle graphs.

Wilson and Roach (1999) and Wilson *et al.* (2001) present a very realistic model, taking into account all technical restrictions in order to arrive at a commercially usable decision support system. Their approach is based on decomposing the planning process into two sub-processes. In the first process, called the strategic process, they create a rough stowage plan, based on grouping the containers with the same characteristics (size, port, etcetera) and assigning them to blocks of storage space in the ship. These blocks represent a number of storage spaces below or above deck. The stability for this rough stowage plan can be calculated to an acceptable degree. In the second phase, called the tactical planning process, individual containers are then assigned to specific slots, resulting in a detailed stowage plan. In the strategic phase, Branch & Bound is used to assign the containers to the blocks in the ship. In the tactical planning process, packing heuristics together with Tabu Search are used to make the detailed assignment of containers to slots. This two level approach seems to be used in practice, where the carriers make the first rough plan and the terminals the more specific plan. Commercial packages exist for this problem, but according to Van der Ham (2001) they seem to be based on greedy heuristics.

2.3 Berth and crane allocation

There are several categories of ships, with large ocean-going vessels as the most important ones and short-sea feeders and barges as the less important ones. A logical way of assigning arriving ships to the berths available at the terminal seems

to be in First Come First Served order within the respective category. However, this may not be a good strategy, especially when the handling time of a vessel depends on the berth it is assigned to. In container handling, this certainly is the case, since a common used stacking policy is to store the containers to be loaded on the same vessel in a more or less dedicated area of the stack. Hence, it is desirable that vessels berth relatively close to this area. In allocating a ship to a berth, one has to take the available berths into account as well as the available cranes to unload and load the ship. Scientific approaches to this problem can therefore be subdivided into pure berth allocation models and crane allocation models. Below we will discuss the contributions in more detail.

Imai *et al.* (1997, 2001) and Nishimura *et al.* (2001) consider the Berth Allocation Problem (BAP) in different versions. The BAP is the problem of assigning ships to berths, such that the sum of the waiting and handling times of the ships is minimized. Each berth can handle only one ship at a time and the handling time of a ship is berth dependent. The authors consider several versions of the BAP. In the static BAP, it is assumed that all ships have already arrived in the port at the beginning of the planning horizon. In this case, there is no idle time between the ships that are handled at a certain berth. The problem can then be formulated as a two dimensional assignment problem, which can be solved in polynomial time. In Imai *et al.* (1997), the static BAP under multiple objectives is investigated. Not only the sum of the waiting and handling times of the ships are minimized, but also the ships dissatisfaction that arises from the berthing order. Although it is assumed that all ships are present at the beginning of the planning period, dissatisfaction may arise since the optimal berthing order with respect to the waiting and handling times, may not correspond with the order in which the ships arrived. The multiple objectives are handled by using the weighting method, which reduces the multiple objectives into a single one. By varying the weights, the set of noninferior berth allocations is identified.

In the dynamic BAP (Imai *et al.* (2001)), the ships arrive during the planning period and as a result, there may be idle time between two consecutive ships serviced at a berth. As a result, the problem cannot be solved in polynomial time anymore. Based on a Mixed Integer Programming formulation, Imai *et al.* (2001) develop a Lagrangian relaxation, which is solved using the subgradient method. Another extension of the model is considered in Nishimura *et al.* (2001). In this paper, the authors consider the dynamic BAP in which it is allowed that more than one ship at a time is handled at a berth, as long as the total length of the ships is smaller than the length of the berth. Moreover, they include the additional constraint that a ship can only be served at a berth if its draft is less than the water depth of the berth. The

problem is then formulated as a non-linear programming problem, which is solved using a genetic algorithm. Lim (1998) also considers this problem and formulates the problem as a restricted version of a two-dimensional bin-packing problem. After proving NP-completeness he develops a heuristic using a graph theoretic model. Finally, Preston and Kozan (2001*a*) consider berth allocation such that the ships are close to the place where the containers to be loaded are stacked. They apply genetic algorithms to solve the berth allocation problem, with the ships turn around time as objective.

Next to these analytical approaches, there have been several simulation studies that employ decision rules for the dynamic allocation problem. We mention Gambardella *et al.* (1998), Lai and Shih (1992), Legato and Mazza (2001), Silberholz *et al.* (1991) and Yun and Choi (1999).

At a slightly more detailed level, the assignments of vessels to berths can be done by taking into account the assignment of the quay cranes to the vessels. Obviously, the number of quay cranes assigned to a vessel, influences the handling time of the vessel. This results in the so-called crane scheduling problem, which has been investigated by Daganzo (1989) and Peterkofsky and Daganzo (1990).

In the static crane scheduling problem (Daganzo (1989)), a number of quay cranes have to serve a number of ships such that the costs of delay of the ships is minimized. Both the berth and the ships are represented by slots. A slot corresponds to a hold of a ship. Each ship requires a certain number of adjoining slots at the berth and each hold can be handled by a limited number of quay cranes (usually 1). Moreover, it is assumed that at time zero, all ships to be handled have arrived at the berth. A Mixed Integer Programming formulation is given for this problem, which is only solved for small instances by enumeration. Furthermore, a heuristic procedure is designed which is based on some crane scheduling principles. In a companion paper by Peterkofsky and Daganzo (1990), an advanced Branch & Bound algorithm is presented in order to solve real-life instances.

The static crane scheduling problem can be generalized to allow for different arrival times of vessels and capacity restrictions on the berth (queuing of vessels). The proposed solution methods for the static case (Daganzo (1989)), can be modified to be used for the more general cases.

2.4 Container loading: scheduling and control of handling equipment

The loading of containers is done according to the stowage plan (see Section 2.2). That is, each container should be loaded at the position in the ship that was determined by the stowage planning. Moreover, from the quay crane scheduling (see Section 2.3) it follows which quay crane handles which holds of the vessel.

The loading process of a container ship consists of the retrieval of the containers from the stack, the transport to the quay and the loading of the containers by the quay cranes into the vessel. The unloading operation is the same but reversed. However, the unloading of a vessel is far less complex than the loading, since during the loading, the stowage plan has to be respected. Hence, the order in which the containers arrive for loading at the quay cranes is important. For containers that are unloaded, the order of unloading is not relevant.

Looking at container terminals worldwide, there are three commonly used handling systems. The first system uses stacking cranes for the retrieval of containers from the stack. These stacking cranes can either be rubber-tired or rail-mounted. In case of rubber-tired cranes, the cranes can usually switch from one stacking lane to another, depending on where the containers for a specific vessel are located. Although this implies more flexibility, changing from one stack lane to another is time consuming and is therefore not done so often in practice. After a stacking crane has retrieved the container from the stack, it loads the container onto a transport vehicle, usually a terminal truck or an automated guided vehicle. Usually such a vehicle can carry one or two containers. However, some terminals use so-called multi-trailer systems, capable of transporting 10 TEU. The terminal vehicle drives to the proper quay crane, which lifts the containers off the vehicle. At most terminals, the stacking cranes and the terminal trucks are manually operated. Several terminals of ECT in Rotterdam however, have automated equipment. In theory, this does not make a lot of difference in the scheduling and control problems that arise, although the actions of drivers can never be completely coordinated. On the other hand, automated systems are less flexible and lack drivers experience and insight. Therefore, detailed and efficient scheduling is much more crucial to achieve satisfactory performance at these automated terminals.

An alternative system for the storage and transport of containers is the use of straddle carriers. Straddle carriers combine the properties of a crane and a vehicle. Hence, they are able to retrieve the containers from the stack and deliver them at the quay cranes. Note that since these vehicles can load and unload the containers themselves, they never have to wait for a crane (provided that there is sufficient

buffer space at the cranes). Based on these arguments, the straddle carrier seems to be the preferable terminal equipment. However, straddle carriers are very expensive and much more unreliable than a system of stacking cranes and terminal vehicles. Moreover, straddle carriers need more space in the stack to operate.

The third possible system, stacking all containers on chassis, is not used that much anymore. Then only terminal trucks are required to drive the trailers to the quay cranes where they are unloaded. Since each container is stored on a chassis, a lot of storage space is required. On the other hand, each container is directly accessible, which makes such a terminal easier to operate.

The choice of handling system is typically a strategic issue since the handling equipment is used for a large number of years. However, often the geographic location of a terminal determines which system will be used. For instance, due to extreme shortage of storage space, most Asian terminals like Hong-Kong and Singapore use a system of stacking cranes and terminal trucks, since stacking cranes allow for higher stacking, compared to a system of straddle carriers, which limits the stacking height to only 3 or 4 containers. On the other hand, if storage space is widely available, for instance in the USA ports, a system using trailers will be preferred because of the relative ease such a system can be operated. In Europe the situation is diverse. Although the stacking height is usually relatively low (2 to 4 containers high), both straddle carriers (Hamburg, Bremen, Rotterdam and Antwerp) and automated stacking cranes combined with AGVs (Rotterdam) are used. Within this context of deciding which handling system to use, we mention the work of Vis *et al.* (2001*b*) in which a simulation study is done comparing these two systems. They claim that on average, the number of straddle carriers required (for transport of containers only, not for stacking) is about 30 percent less than the number of AGVs required. This would favor the use of straddle carriers. However, when deciding on which system to use, also costs and reliability have to be taken into account, which usually do not favor straddle carriers.

In the next subsections, we will discuss various scheduling problems of handling equipment. These problems all appear at the operational level. Obviously, the purpose is to use the available equipment in the most efficient way. However, the overall objective is to minimize the in–port time of the vessels. Since the operating costs of a vessel are very large, its economies of scale can only be realized whenever it is actually transporting containers, not whenever it is waiting at a terminal.

2.4.1 Scheduling of stacking cranes

Kim and Kim (1997, 1999*b*) consider the optimization of the routing of a single stacking crane. Such a stacking crane operates on a stacking block, which is again divided into a number of bays. The stacking 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 stacking 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 stacking crane, the work schedule is assumed to be 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 stacking 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 on which an optimization algorithm using Dynamic Programming is based. In a related paper, Kim and Kim (1999*c*) use a Beam Search algorithm to solve the model. The performance of the algorithm was tested using 360 sample problems. Moreover, for several smaller instances, the Beam Search solution is compared with the optimal solution. The authors report an average difference in travel distance of about 15 percent.

A comparison between the use of a stacking crane and a straddle carrier to retrieve containers from the stack, is made by Vis (2002). Both the stacking crane and the straddle carrier operate on a block, which consists of several rows of containers. The difference between the equipment is that the stacking crane can switch to another row at any location, where the straddle carrier always has to drive to the end of a row to switch. For both cases, an algorithm is presented which minimizes the empty driving distance. An extensive computational study is done to compare the performance of both types of handling equipment.

Murty *et al.* (2000) consider the problem of deploying rubber-tired stacking cranes during several work shifts. They consider a terminal in which there are stacking blocks that have a certain workload in every shift. The stacking cranes are allowed to switch from one block to another. However, the number of switches is restricted to one per stacking crane, since the movement of such a crane from one block to another is time consuming and moreover, the roads between the blocks are obstructed for the terminal trucks. The objective is to assign the stacking cranes such that the amount of workload that cannot be finished within a shift is minimized. Next to a Mixed Integer Programming formulation, also an approach based on Lagrangian

relaxation was developed. However, one of the shortcomings of these models is the restrictive assumption that all work is available at the beginning of each period, where in practice, the work "arrives" during the period. Moreover, terminal operators considered the approaches too complicated for practical implementation. Therefore, an alternative heuristic approach was developed, based on expert judgement of crane supervisors.

2.4.2 Scheduling of AGVs

Although AGV systems also appear in flexible manufacturing systems (Egbelu and Tanchoco (1984)) and warehouses (Van der Meer (2000)), the scheduling and control problems that appear at automated container terminals are usually more complicated. For instance, deadlocks and congestion are major problems. This is mainly due to the large number of vehicles that is deployed and the large size of the vehicles themselves. For instance, the number of AGVs working at the loading operation of a single vessel is about 30, while their length is some 17 meters and width 2.5 meters. Moreover, the scheduling of the AGVs also interferes with the schedules of the quay cranes and the stacking cranes, which is a complicating factor (see also Subsection 2.4.4).

The scheduling of AGVs is considered in Chen *et al.* (1997) and Kim and Bae (1999). Chen *et al.* (1997) consider the dispatching problem of AGVs in an automated container terminal. To simplify the analysis, they assume that stacking cranes are always available to load or unload the AGVs. For the case of a single quay crane, which is either loading or unloading, they derive a greedy algorithm for assigning the containers to the AGVs, which is shown to be optimal. For the case of multiple quay cranes, the greedy algorithm does not necessarily give the optimal solution and is therefore used as a heuristic. The algorithm was tested within a simulation model in order to investigate its effectiveness and robustness. Moreover, a queuing model was developed. Given the use of the greedy dispatching policy, this model gives insight into the effects on the performance of using more AGVs, more cranes, etcetera.

Another approach to the scheduling of AGVs is given by Kim and 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 quay crane and the given event times, they reduce the dispatching problem of AGVs to an assignment problem. Next, for the case in which the event times cannot be met, a heuristic is developed.

Dispatching rules for AGVs in a more general context were investigated by Egbelu and Tanchoco (1984) and Van der Meer (2000). Although dispatching rules seem to

be generally applicable, the specific context of an automated container terminal does not allow for a straightforward implementation, as will be shown in Chapter 5 of this thesis. Hence, to apply these dispatching rules at an automated container terminal, like the ECT terminals in Rotterdam, they need to be adjusted, as will be discussed in Chapter 5.

Vis *et al.* (2001*a*) consider the problem of determining the number of AGVs required at a 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.

2.4.3 Traffic control of AGVs

Next to the scheduling of AGVs, their control is another main issue. Although AGVs at container terminals are free ranging, they do follow fixed paths for their routes. Moreover, since the orientation of containers (the direction of the doors) is prescribed both in the ship and in the stack, the AGVs have to make complex turns, which require lots of space. The traffic control mechanism should guarantee that the AGVs do not collide. For instance, if the paths of two AGVs cross, the control mechanism should determine which AGV goes first and which AGV has to wait. Or whenever there are two parallel curves, only one of them can be used by an AGV at a time, since the other is then blocked.

The control system can be either central, so the control is done by a central system, or distributed, which implies that the AGVs and certain areas possess a form of intelligence. The current generation of automated container terminals in Rotterdam uses a central control system. However, Evers and Koppers (1996) propose a distributed control system using a hierarchical system of semaphores. Roughly speaking, the semaphores act as traffic lights that control the number of AGVs that enter a zone. The advantages of the approach are the flexibility in modeling a transportation system and the little communication that is needed to control the AGVs (the AGVs communicate only with local areas). Although one of the purposes of the traffic control system should be to make the AGV travel times more predictable, they are still stochastic, making the dispatch a difficult scheduling problem, since sending the AGVs earlier on their way causes even more traffic and sending them late may cause them to arrive too late. Finally, the strict loading order at the quay cranes also causes problems. Since the AGVs should arrive in a certain order at the quay cranes and overtaking in the queue (AGV track) in front of the quay crane is not possible, the traffic control system should guarantee that the AGVs enter the queue in the right order. Taking into account the specific difficulties of the AGV system at

a container terminal, Duinkerken and Ottjes (2000) have performed extensive simulation studies in order to investigate the performance of such an AGV system under intelligent traffic control.

2.4.4 Integrated scheduling of stacking cranes and AGVs

The main loss of performance at a container terminal that uses stacking cranes for the retrieval of containers and AGVs (or terminal trucks) for the quay transport, is the fact that the schedules of the various equipment are uncoordinated. Hence, empty AGVs are waiting for stacking cranes to load the next container and vice versa. In the worst case, even deadlock situations may appear. Therefore, we propose to schedule quay cranes, stacking cranes and AGVs in an integrated way. The models that will be presented in this thesis, explicitly take into account the topology of the terminal. In Chapter 4 of this thesis, which is based on Meersmans and Wagelmans (2001*b*), we consider the case in which all AGVs pass a common point after unloading at the quay cranes (the current situation at ECT's Delta terminals). It is shown that the order in which the AGVs start at the common point to pick up their next container, completely determines the remaining part of the schedule, also for the stacking cranes. Using this result, we present a Branch & Bound algorithm that uses several combinatorial lower bounds to cut off unpromising parts of the search tree. These lower bounds turn out to be very effective, since the algorithm performs well not only on small instances, but also on large real-life problems. Moreover, based on this Branch & Bound algorithm, a heuristic Beam Search algorithm is developed. In most cases, this Beam Search heuristic gives a solution that is within 5 percent of the optimum, requiring far less time than the Branch & Bound algorithm. A dynamic version of the Beam Search algorithm is discussed in Chapter 5 of this thesis. In this chapter, based on Meersmans and Wagelmans (2001*a*), we also propose a number of dispatching rules. In an extensive computational study, both the performance of the dynamic Beam Search algorithm and the performance of the dispatching rules is investigated.

The case in which the AGVs do not pass a common point after unloading is discussed in Chapter 6, which is based on Meersmans *et al.* (2001). In such a more general layout, AGVs may, for instance, take shortcuts directly after unloading at the quay crane. For the resulting integrated scheduling problem, we present a Mixed Integer Programming formulation for which several classes of valid inequalities are derived. These inequalities turn out to be very effective in reducing the computational effort to solve the model.

2.4.5 Scheduling of straddle carriers

An alternative handling system for the retrieval and transport of containers by stacking cranes and AGVs (terminal trucks), are straddle carriers. Unlike AGVs, straddle carriers are able to pick up and drop off containers themselves. As a result, straddle carriers can be used for both the retrieval of containers from the stack and the transport to the quay cranes. However, in order to overcome the dependency of the AGVs on the stacking cranes, straddle carriers are also often used for transport purposes only.

Böse *et al.* (2000) consider the dispatching of straddle carriers for several loading and unloading operations (vessels) at the same time. They use a genetic algorithm to minimize the delays of container transfers to the quay cranes. Next to a situation in which there is a static assignment of straddle carriers to quay cranes, they also investigate the effects of a dynamic assignment, i.e., the pooling of the straddle carriers which are working on different (un)loading operations. Computational tests are done for multiple loading and unloading operations and for different layouts of the terminal. The results show that a dynamic assignment can lead to significant productivity improvements.

Work done by Winter (1999) concentrates on 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 quay crane, a sequence of categories to be loaded is then given. Hence, during the loading of the vessel, 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. For the case of a single quay crane, a Mixed Integer Programming model is given, which provides optimal or near optimal solutions in short time. For multiple quay cranes a best-fit heuristic is proposed.

Kozan and Preston (1999) consider the optimization of container transfers using straddle carriers (called yard machines). The objective is to minimize the time the vessels stay in the port. Their model explicitly deals with the reshuffling of containers by introducing setup times. These setup times depend on the order in which all containers on top of a specific container are handled. The model presented is solved by using genetic algorithm techniques. In a companion paper, Preston and Kozan (2001*b*) solve the model using a tabu search heuristic. Computational results show that in most cases, the tabu search gives better solutions than the genetic algorithm, in far less time. The tabu search heuristic is also used investigate the effects of

different stack configurations on the time needed to transfer all containers to the quay.

2.5 Stacking

The stack allows the various transport activities to occur independently of each other. Would there be no stack then every container ship arrival would have to be followed by direct unloading to inland barges or to trucks. This would result in a very complex logistical operation because all arrivals would have to be coordinated. Hence, congestion and delays would be inevitable. As discussed in Section 2.1, different stacking systems exist. In this section we will only consider stacking in piles on the ground, which is done at most terminals.

The determination of the stack capacity is a major design problem of a terminal, as the stack occupies much costly land. Stacking many containers on each other may be advocated, but the expected number of reshuffles increases sharply with the stacking height. A reshuffle is an unproductive move of a container which is required to access another container that is stored beneath it. This is illustrated in Figure 2.3: container 1 is directly accessible, container 2 demands one reshuffle.

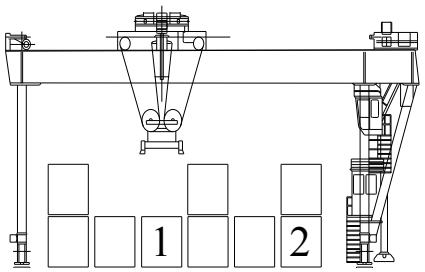

Figure 2.3: Reshuffles

Related to the stack capacity is its layout: how long and wide will it be and how many blocks or lanes will be used? Next, one has to decide in the design what type of equipment will be used for stacking. The two most common types are straddle carriers and stacking cranes. Next to these two, some terminals operate with a stack on wheels, in which all containers are put chassis. Obviously, this requires a large storage area and high investment costs in chassis.

Quite often stacks are separated into import and export parts. Import containers usually arrive in large container ships and hence, in a somewhat predictable way. Yet they are likely to depart in an unpredicted order, so, they cannot be stacked

that high. Export containers may arrive somewhat randomly, but their departure is usually connected to a ship and hence, they can be stacked in a much better way.

At the tactical level, one has to decide on allocation of parts of the stack to certain activities, for instance, the (un)loading of a certain ship. At an operational level, one has to decide on which container to stack where in order to avoid reshuffles as much as possible. Note the close relation to the stowage planning as discussed in Section 2.2. Also in the stowage planning, containers are stored on top of each other, such as to minimize the number of reshuffles. However, the stacking problem is more difficult since there can be uncertainty about which container will be needed first. For import containers, this uncertainty exists because trucks arrive more or less randomly to pick up a specific container. For export containers, it is usually known with which ship they depart. However, the stowage plan is usually available only shortly before the loading starts and thus, most containers are already stored in the stack. This leads to reshuffles which result in a slowdown of the loading operation. In case the stowage plan is available earlier, the containers in the export stack may be remarshalled. This results in an "ideal" stack, i.e., the containers are stacked in the order of loading. Hence, there is less handling during the loading operation of the vessel. Kim and Bae (1998) describe a two–stage approach in order to minimize the number of containers to be moved and to do so in the shortest possible traveling distance. Although such a re-marshalling approach seems very attractive, it is not often possible to do so. As already mentioned, the stowage plan is only known shortly before the loading operation starts. Moreover, building up this "ideal" stack also requires additional storage space which may not be available.

Stacking problems can be dealt with in two ways: simplified analytical calculations or detailed simulation studies. The first gives insight into the relationships between the various parameters on a more abstract level. The second can go in much more detail, with the negative by-effect that it is time consuming and only few people really understand its ins and outs. No comprehensive stacking theory exists today, and a good stack design not only depends on local space conditions but also on the information characteristics of the ingoing and outgoing flow of containers which may vary from port to port. Examples of both approaches are given below.

Sculli and Hui (1988) were among the first to develop yardsticks for the relation between stacking height, utilization (or storage space needed) and reshuffles by applying a comprehensive simulation study. Taleb-Ibrahimi *et al.* (1993) discuss this relation for export containers both at a long-term scale and at an operational scale. They discuss dynamic strategies which store early arriving containers in a rough pile until a certain date, after which all containers for a ship are put in a dedicated storage area (usually close to the berthing place of the ship). The developed procedures

calculate the storage space needed as a function of the stacking height. De Castilho and Daganzo (1993) continue these studies with the stacking of import containers. They consider two strategies, one that keeps stacks of the same size versus one that segregates the containers on arrival time. A slightly more detailed discussion resulting in tables and yardsticks (looking at stacking blocks with bays of similar sized containers), both analytically and by simulation, was given by Kim (1997). Kim and Kim (1998) extended these studies by also taking the number of stacking cranes into account. They developed a simple cost model for optimizing this number using analytical approximations for the various performance measures.

Segregating space allocation strategies of import containers were studied by Kim and Kim (1999*a*). In segregation strategies, stacking newly arrived containers on top of earlier arrived containers is not allowed. Spaces are thus allocated for each arriving vessel. They study cases with constant, cyclic and varying arrivals of vessels.

An empirical statistical analysis of the actual performance at a Taiwanese container terminal was provided by Chen *et al.* (2000). The number of shift moves was related to the storage density, the volume of containers loaded and the volume of containers discharged both for stacking crane blocks and straddle carrier blocks.

More popular discussions of the effect of different stacking heights appeared in the professional journal Cargo Systems. We mention Ashar (1991), De Castilho (1992) and Watanabe (1991).

The relation between space allocation and the transport of the containers using vehicles, is studied using combinatorial optimization by Demir *et al.* (1998). They look for strategies for dispatching the vehicles to minimize the total time to download all the containers from the ship. They prove NP-hardness of the problem and present heuristics for which absolute and asymptotic worst-case performance ratios are derived. Murty *et al.* (2000) consider storage space assignment, taking into account congestion that may appear if too many terminal trucks have to deliver (pick up) containers in the same area of the stack. A two stage method is proposed. First, the arriving containers are assigned to blocks in the stack. Next, each container is put in the best position in its block, taking into account possible reshuffles. Stacking policies are investigated by Duinkerken *et al.* (2001), who use a detailed simulation model that not only models the stack, but also the quay transport in an automated container terminal.

Decision rules using weight groups for locating export containers were derived and validated through dynamic programming by Kim *et al.* (2000). Weight is a useful criterion since heavy containers are usually stored deep in the ship (see also Section 2.2).

In an extensive simulation study, Dekker *et al.* (1999) consider the effect of cre-

ating groups of exchangeable containers in the stacking. They show that having such groups not only decreases the number of reshuffles considerably, but also the workload during the loading of the vessel can be much more evenly distributed in automated systems. Unfortunately, such exchangeable groups are only likely to be created for export containers, not for import containers, since import containers have many different destinations (individual customers). Hence, for the latter, stacking remains a severe problem.

2.6 Overall terminal studies

Container terminals can grow out into large complexes with several marine terminals, empty depots, container repair centers, rail terminals, barge service centers, etcetera. Studying a whole complex and the relation between all the components is therefore the last aspect of container terminals we like to discuss. The literature on this aspect is somewhat scattered and we like to distinguish the following groups. First of all, there is the design of an overall terminal complex. Next, there is the study of the various other interfaces of a terminal with other modalities, like a rail terminal, and finally, there is the study of all the transport streams in and between the various parts of the terminal complex.

2.6.1 Overall container terminal design

Consultants often apply spreadsheet models as little detailed information is known and quick answers have to be provided using yardsticks and experience. Few of these approaches have been published. A more detailed model is discussed in Van Hee *et al.* (1988) and Van Hee and Wijbrands (1988). The authors describe a decision support system (DSS) for capacity planning at container terminals. The model uses several analytical models that are integrated within a single DSS. The DSS contains models for computing service times of stacking cranes, throughput of quay cranes and the behaviour of the whole terminal.

Another contribution on tactical level (months) capacity planning is by Zaffalon *et al.* (1998), which give a network model for resource allocation of terminal equipment over a number of shifts. A detailed simulation model is presented that is used to validate the resource allocation.

Wong *et al.* (1983) present a generic simulation model for terminal layout and the determination of the type and number of handling equipment. They apply it to a hypothetical terminal. Generic yardsticks for handling productivity at a container terminal are given by Al-Kazily (1983).

2.6.2 Rail terminals

Productivity at rail terminals is studied by Kozan (1997) and Van Zijderveld (1995). Using probabilistic calculations, Kozan (1997) determines the performance of various handling activities at a rail terminal. A much more complex study on a rail terminal, using simulation, was carried out by Van Zijderveld (1995). He considered loading and unloading of trains at a rail terminal with two concepts: one in which trains stayed shortly at a terminal and were shunted to a shunting yard quickly thereafter, and a rail service center where trains stay relatively longer, but without a shunting yard. The point is that when multiple trains are handled at the same time, productivity can be improved.

2.6.3 Landside interface

A detailed simulation model for straddle carriers that have to perform landside tasks is described in Behera *et al.* (2000). The authors model the route network of the terminal and take into account the delays that may appear if too many straddle carriers operate in the same area.

Sinclair and Van Dyk (1987) consider the problem of pick up and delivery of containers by road transport from a container depot. The problem differs from the well known pickup and delivery problem in the sense that trailers are used for the transport of containers. Whenever a truck arrives at a client or the depot, it can either leave only the container, or both the container and the trailer. The problem is solved using a two stage heuristic procedure.

2.6.4 Short sea shipping

A new terminal concept for short sea shipping is described by Ottjes and Veeke (1999) and Veeke and Ottjes (1999). In this concept, containers are stored on frames, which are picked up by AGVs who drive into the vessel. This concept can be compared with the traditional "roll on roll off" systems that exist nowadays. Simulation was used both for proving the feasibility of the concept and for getting insight into the dimensioning of such a terminal.

2.6.5 Inter terminal transport

Transport in and between several terminals can be time–critical, as incoming sea containers may have to be loaded on a train during the time window the train is on the terminal. Hence, a good layout is essential for efficient operations. The transport may be done with single trailers, multi-trailer systems or straddle carriers. Few

papers have addressed these issues in a comprehensive way and in fact only scattered results are available so far. Therefore, we just mention all individual papers.

Kozan (2000) and Kozan and Preston (1999) consider container transfers at a (multimodal) terminal complex. They model the problem as a large network with a mathematical programming formulation which is then solved with genetic algorithms. Some of the elements within this formulation are expected throughput times based on probabilistic calculations. The technique is applied to the Brisbane Multimodal Terminal and identifies and solves bottlenecks using sensitivity analysis.

In Kurstjens *et al.* (1996) and Ottjes *et al.* (1996), a study is described on the best transportation system for inter terminal transport at the Rotterdam Maasvlakte (where ECT resides). Three transportation systems were compared, viz. a multi-trailer system capable of transporting 10 TEU, an AGV system using existing AGVs and a new system using so-called automated lift vehicles (ALV), comparable to an automated straddle carrier. From the study it appeared that all systems have different characteristics and that there was quite a difference in their logistic performance. The first two systems need cranes to put their containers on them, which resulted in many bottlenecks in the handling, while the ALV could put its container at its destination on the ground and continue with another task. Accordingly, in the scenario studied, about 65 ALVs had the same logistical performance (the same number of containers transported within prescribed time windows with about the same number, 1%, too late) as 130 AGVs. As the multi-trailer systems had (manned) trucks which can be decoupled from their trailers, some 22 trucks were needed with some 130 trailers (each capable of carrying 10 TEU) to achieve the same performance. As the outcomes depended strongly on the way the transports were planned and controlled, quite a large part of the study concentrated on these aspects. These problems turned out to be much more difficult than ordinary vehicle scheduling problems, because of the interfacing with cranes to finish tasks and the many stochastic elements.

Steenken (1992) and Steenken *et al.* (1993) consider the routing of straddle carriers that have to perform internal and landside moves. Each move has to be carried out before a certain due date. The problem is modeled as a Multiple Rural Postman Problem and two kinds of heuristics are used for solving it. The first heuristic comes from the field of printed circuit board assemblies. The second class of heuristics are dispatching rules like the Earliest Due Date (EDD) and Shortest Processing Time (SPT) rule.

2.7 Conclusions

Container terminals are very specific from a material handling point of view, because of the special characteristics of both the containers and the handling equipment. Terminals have become increasingly important and more and more scientific literature is devoted to them. This is even more true for the automated terminals which are being established to cope with the increase in labor costs. The further increase in ship sizes makes a productivity improvement in container handling more important and therefore more research is to be expected. In this chapter, we have discussed the relevant decision problems within container handling and we have tried to classify the existing scientific approaches.

Operations Research has made valuable contributions for container terminals. The techniques employed vary from Mixed Integer Programming formulations, queuing models and simulation approaches. The first are used for planning under several constraints. The second are used to model and understand some of the queuing phenomena at terminals. The latter are used to gain more specific insight in case many operational aspects are included. Yet they provide less generic insights and the many details included make it difficult to claim generality of the results.

Our experience with ECT's automated terminals shows that the real problems are extremely complex and that existing approaches need much more development to help in taking the very costly decisions.

Chapter 3

Integrated scheduling of handling equipment: problem description

In this chapter, we give a description of the scheduling problem of handling equipment at an automated container terminal. We will first discuss the layout and the handling equipment used at such a terminal. Next, in Section 3.2 we will focus on the (un)loading operation of a container vessel. In Section 3.3, we will shortly discuss the detailed simulation model of an automated container terminal that we developed, which is used for testing of the scheduling algorithms we discuss in this thesis. Next, in Section 3.4, we describe the scheduling problem of terminal equipment in more detail. Finally, in Section 3.5, we will motivate our solution approach.

3.1 Automated container terminals

The first of a series of automated container terminals was put into operation in 1993, at the ECT peninsula in Rotterdam, The Netherlands. This terminal uses automated stacking cranes (ASC) for the storage and retrieval of containers in the stack. Moreover, the transport of containers from the stack to the quay cranes (QC) is done using automated guided vehicles (AGV). Only the quay cranes are still manually operated. So far, Rotterdam is the only port in the world in which automated container terminals are in operation. However, currently, an almost identical terminal is constructed in Hamburg and plans for such a terminal are made in the ports of

Singapore and Antwerp.

Figure 3.1 gives an overview of one of ECT's automated container terminals. We will now discuss briefly the main elements of this terminal.

Figure 3.1: Overview of one of ECT's automated container terminals (photo: courtesy of ECT)

3.1.1 Stack and ASCs

The stack covers a major part of the terminal and is used for temporary storage of containers while they change from one modality to another. For instance, a container arrives by sea going vessel and leaves by truck or train. The stack is divided into 27 stacking lanes. In each lane, containers can be stored in six rows. Each row provides 42 TEU of ground positions. On each ground position, two or three containers can be stored on top of each other. At the endpoints of the stacking lanes, there are so-called transfer points. At the waterside transfer points, the AGVs pick up (deliver) the containers for further transport to the quay. At the landside transfer points, straddle carriers pick up (deliver) containers in order to load them on trucks.

The storage and retrieval of containers within a stacking lane is done by a single stacking crane (ASC). This ASC is fully automated and can work only on a single stacking lane since it is rail-mounted. Figure 3.2 shows several stack lanes, each with its dedicated ASC.

Figure 3.2: Stack lanes with ASCs (photo: courtesy of ECT)

The movements of the ASC necessary to retrieve a container from the stack consist of hoisting, driving and positioning. These movements are performed sequentially. On the landside, the ASC puts the containers on the ground, since they are picked up by a straddle carrier which is able to pickup the container itself. On the waterside, the ASC always has to wait until an empty AGV has arrived.

3.1.2 AGVs

The AGVs are used for the transport of containers from the stack to the quay and back. The AGVs used at the ECT terminals have a length of about 17 meters and a width of about 2.5 meters and are able to carry only a single container of either 20,

40 or 45 ft. The vehicles have a rather simple structure and always require a crane to load or unload a container. This is illustrated in Figure 3.3.

Figure 3.3: Loading of an AGV by an ASC (photo: courtesy of ECT)

The AGVs are equipped with sensors in order to locate obstacles at an early stage. The control of the vehicles is centralized, using a block system. This means that each AGV claims a number of blocks. These blocks are then not accessible by other AGVs. In this way, collisions are prevented.

The AGVs operate only in the area between the stack and the quay and follow predefined tracks. At the current automated terminals of ECT, the AGVs drive clockwise, in a loop layout. Moreover, after they are unloaded by the QC, they all pass a common point before they can make a turn towards the stack. This common point is situated after the last quay crane that is operating on the vessel. The middle area between the tracks at the waterside and the stack side is reserved for storing the hatch covers of the vessels. Future container terminals may use AGV tracks which are not in a loop. For instance, the AGVs may cross the middle area, directly after unloading at the QC, i.e., the AGVs take shortcuts between the QCs.

Due to the large number of AGVs that are deployed (at most 50), there can be a

lot of interference between the various AGVs. That is, it is likely that AGVs have to wait for other AGVs to pass before they can enter a track. This makes the driving times of the AGVs somewhat unpredictable.

3.1.3 Quay Cranes

The QCs are the only manned equipment at the automated container terminal. They load and unload containers from the vessels at a rate of about 40 to 50 containers an hour. Obviously, their performance depends on the performance of the AGVs and ASCs, which have to make sure that the right containers arrive at the right time at the quay.

The QCs have a spreader which is attached to the container. The spreader is moved by cables. Because of this construction, it can be difficult to position a container in the ship, since the wind may move the container. Therefore, the handling times on the QCs are rather stochastic.

The QCs handle the containers according to a predefined list, which follows from the stowage plan and the crane split. More details are given in the next section.

3.2 Loading and unloading operation of a vessel

After the ship has moored at the quay and the containers destined for the port are unloaded, the loading operation starts. 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 information on stowage planning, we refer to Section 2.2.

Modern container vessels have up to 25 holds in which containers have to be loaded. The size of these vessels allows for the deployment of 3 to 6 QCs at the same time, where each QC handles a number of holds. The crane split, i.e., the assignment of holds to QCs, and the order in which a QC handles the holds, is a problem in itself and is assumed to be given. We refer to Section 2.2 for more information. Within a hold, the containers are loaded in a fixed order. In practice, this is either row-wise or column-wise. This is illustrated in Figure 3.4, which gives a cross-section of the vessel. Hence, since we know the order in which the QC handles the holds and since within a hold 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.

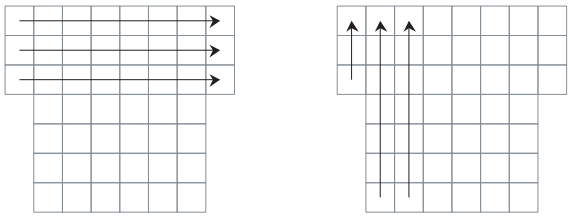

Figure 3.4: Two common loading strategies

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, which lifts the container off the AGV and loads it into the vessel. The handling of a container that is unloaded from the vessel is done in reverse order.

The unloading operation of a vessel is far less complex than the loading operation. Per hold, the containers are unloaded by the QCs and transported by AGVs 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. However, in case of a container to be loaded, weight, destination and other characteristics do matter, since the stowage plan has to be respected. This makes the operation more complex since the individual containers have to be delivered in the right order to the QCs. Therefore, we will consider only the loading of containers in this thesis, since this is the most critical and complex operation in the handling process of a vessel.

As already mentioned in Chapter 1, the operating costs of seagoing vessels are very high, over $1,000 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.

3.3 A simulation model

In this section, we will discuss the main ingredients of a detailed simulation model of an automated container terminal. For more details on the simulation model, we refer to Meersmans *et al.* (1999). The simulation model was developed in close cooperation with ECT, the operator of the automated terminals in Rotterdam.

The simulation model was set up to investigate the performance of scheduling algorithms for terminal equipment. To this purpose, the model can simulate the unloading and loading operation of a vessel. Next to that, the model also allows for simulation of additional container moves related to landside activities, i.e., stacking crane moves to store and retrieve containers that arrive by landside modalities like trucks or trains.

Another purpose for which the simulation model can be used, is the investigation of more strategic or tactical issues. For instance, one may like to determine the number of AGVs that are required to achieve an acceptable performance of the terminal. Obviously, this number also depends on the algorithms that are used for scheduling the equipment. But given a certain algorithm which is used, the effect on the performance of adding several additional AGVs can be easily determined using the simulation model. Moreover, the simulation model is set up in such a way that it allows for specifying different kinds of layouts of the terminal. Hence, it is possible to investigate the effects of different layouts on the overall performance. Note however, that a drastic change in layout may require the use of other algorithms for scheduling the handling equipment. For instance, the algorithm which we will describe in Chapter 4 of this thesis, cannot be applied in the situation that is considered in Chapter 6.

Since the stack is modeled in detail, the simulation model can also be used to investigate the performance of stacking rules, for instance, the effect of a stacking rule on the number of reshuffles that are required during a loading operation.

The simulation model consist of a number of components, which we will discuss shortly in the following subsections.

3.3.1 Generation of a vessel

The first main element of the simulation model is the generator of vessels. Essentially, a vessel consists of a number of containers to be unloaded and a number of containers to be loaded. A vessel is divided into a number of holds. Each hold contains a number of containers below and above deck. This distinction is made since containers below deck require a longer handling time by the QC than containers above deck.

The containers that are unloaded from the ship are all considered to be the same. That is, they are all import containers and the position in the stack where they will be stored is not important (with respect to weight considerations, etcetera). For containers that will be loaded however, the position in the ship does matter, i.e., the stowage plan (see Section 2.2) has to be respected. For the moment, we use the model in such a way that each container is treated as an individual. So, each container belongs to a specific position in the ship and there is no interchanging possible of containers with the same size, destination and weight. However, the model does allow for such interchanging, which results in categories of containers.

The number of containers to be loaded and unloaded can be specified by the user. These containers are randomly divided over the number of QCs, which is also specified by the user. It is assumed that each QC handles a fixed number of holds of the vessel (the holds are equally divided over the QCs). The containers are then divided randomly over the holds (below and above deck) that the QC handles. The holds are handled by the QC in a fixed order. So, in case of containers that have to be loaded, it starts with handling hold 1 below deck, then hold 1 above deck, then hold 2 below deck, etcetera. In this way, we get for each QC the sequence of containers to be loaded. As already mentioned in Subsection 3.2, this sequence is kept fixed.

3.3.2 Generation of a stack configuration

Given the containers to be loaded by the QCs (see the previous section), we have to assign these containers to positions in the stack. The dimensions of the stack, i.e., the number of lanes, the number of rows per lane, the number of ground positions per row and the maximum stack height, can be specified by the user. In the stack, we do not distinguish between containers of different size: we assume a "uniform" container of 30 ft. (the average of 20 and 40 ft.).

The dimensions of the stack determine the maximum number of containers that can be stored in it. The user can then specify a stack occupancy. Based on this occupancy, the appropriate number of containers is generated. The containers are then randomly assigned to ground positions in the stack, keeping into account that no more containers can be stored on a single ground position than the maximum specified stack height. So, we end up with a pile of containers on each ground position. These containers are all dummy containers and influence the loading process since they may have to be reshuffled.

The matching between the containers to be loaded onto a vessel and the current stack configuration is then done as follows. For each container to be loaded, we randomly draw a position in the stack. That is, we do not only randomly pick a

ground position, we also generate a random level (between 1 and the maximum stack height). We then check if on this ground position a dummy container is stored on the generated level. If so, we replace the dummy container by the container that is to be loaded onto the vessel. If not, we repeat the procedure and pick a new ground position and a new level.

The result of this procedure is a stack configuration in which the containers to be loaded are randomly stored. Obviously, this may result in unproductive reshuffle moves during the loading operation, since (dummy) containers may be stored on top of containers that have to be loaded. The dummy containers that have to be reshuffled during the loading of a vessel are then moved to ground positions where there is no container stored that still has to be loaded onto the vessel. If such a position does not exists the dummy container is moved to a random ground position.

3.3.3 Generation of landside moves

In case there is no separation between the so-called import and export stacks, i.e., incoming and outgoing containers are stored in the same stack, then also containers related to the landside operations have to be modeled. In the simulation, we model these activities only by their claim on the ASC capacity, i.e., the retrieval or storage of containers in the stack by the ASCs. The rest of the landside operations, for instance, the loading of trucks by straddle carriers, is not modeled.

We assume that the landside tasks arrive according to a Poisson process. With equal probability, the task concerns the storage or the retrieval of a container from the stack. In case of storing a container, a random (available) position in the stack is chosen. In case of a retrieval, a dummy container already in the stack is randomly chosen (provided that this is not a container that has to be loaded onto the vessel). Obviously, it may happen that the chosen container also results in one or more additional reshuffle moves. The landside tasks are generated as long as the unloading and loading of the ship has not finished yet.

3.3.4 Terminal equipment

In this subsection, we discuss the detailed modeling of the terminal equipment within the simulation model. For each of the three types of equipment (QC, AGV, ASC), the handling speed of the various operations can be specified. The equipment handles its containers based on a list which is specified by the terminal control (see next subsection). Essential in the modeling of the terminal equipment is the detailed modeling of the dependencies between the various equipment. That is, the transfer of containers from ASC to AGV can only take place if the proper AGV has arrived.

Until that time, the ASC is blocked and cannot continue with its next task. The same happens at the QC, where the AGV has to wait until it is unloaded by the QC before it can move on to its next task.

QCs

The QCs handle their containers according to the loading plan, which results from the vessel generator as discussed in Subsection 3.3.1. Since the handling of containers by the QCs is still a manual operation, the handling times of the containers are not known with certainty. The handling time of a specific container can be estimated based on the position it is assigned to in the ship (see for instance Figure 3.4). Given this estimate, the actual handling time can be shorter or longer. This can be modeled by specifying a distribution which defines the deviation from the expected handling time. In the simulation, any type of distribution for the deviation can be specified. Note that this results in a handling time distribution which is different for each container.

AGVs

The AGVs transport the containers from the stack to the quay and back. The actual path that the AGV follows is not simulated in detail. It is assumed that an AGV picks up a container at the transfer point at the stack and delivers it at the transfer point near the QC. Next, it drives back to one of the transfer points at the stack to pick up its next container. Both operations take a certain handling time. Since there may be interference between AGVs, the handling times are not deterministic. The interference between AGVs is only specified in terms of a deviation of the expected handling time. The distribution of the deviation of the handling time can be specified by the user. The AGVs handle the containers according to a schedule that is specified by the terminal control (see next subsection).

ASCs

Each stack lane has exactly one ASC assigned to it, which performs all tasks within the lane. The operations of the ASCs are modeled in detail. They consist of driving, hoisting and positioning and are carried out sequentially. Each operation is performed with a certain speed, which can be controlled by the user. This results in the handling time for each container, which obviously depends on the exact position (including the height) of the container in the stack. The ASCs perform three kinds of tasks: storing and retrieving of containers related to the (un)loading of a vessel, storing and retrieving of containers related to landside operations, and reshuffling of containers.

The tasks are handled according to the schedule which is determined by the terminal control (see next subsection).

3.3.5 Terminal control

Terminal control is the most important part of the simulation. It coordinates the simulation by providing each piece of equipment with a detailed schedule of the operations it has to perform. For the QCs, the order of the containers follows directly from the loading plan of the ship. At the start of the simulation, terminal control calls the vessel generator to generate an unloading/loading plan and the corresponding lists of containers to be handled by the individual QCs.

Next, given the loading plan, terminal control calls the stack generator in order to match each container to be loaded with a position in the stack.

A planning horizon can be specified by the user. This planning horizon gives for each QC the next C containers that will be loaded or unloaded (note that the planning horizon can also cover the entire sequence of containers of a QC). These containers are then scheduled on the various equipment. For containers that are unloaded, we first have to determine the stack lane in which the container will be stored. This is done randomly. As a result, we then know for every container where it is located in the stack, or where it will be stored in the stack. So, we can then generate the tasks that are related to the (un)loading of each container. Next to the tasks that result from the (un)loading operation of a vessel, there are also ASC tasks generated which are related to landside operations (see Subsection 3.3.3).

Given all the tasks for the various equipment that are available, a scheduling module is then called by the terminal control which returns a detailed schedule, both for the AGVs and ASCs (the order in which the AGVs and ASCs should handle their containers). When constructing a schedule, we use the expected handling times of the containers. For the moment, we assume that the scheduling module is a black box. In the remainder of this thesis, we will focus on algorithms that can be used within this black box in order to schedule the tasks as efficiently as possible.

The schedule returned by the scheduling module is communicated to the modules that model the various pieces of equipment. As time passes, tasks are executed and the information about the realization of the schedule by the various equipment is given back to the terminal control. Moreover, new tasks may appear within the planning horizon. So, after a fixed period of time, the available tasks at that time are rescheduled. The frequency of rescheduling can be specified by the user.

The relations between the terminal control and the other modules of the simulation, and the information flows between them, is illustrated in Figure 3.5.

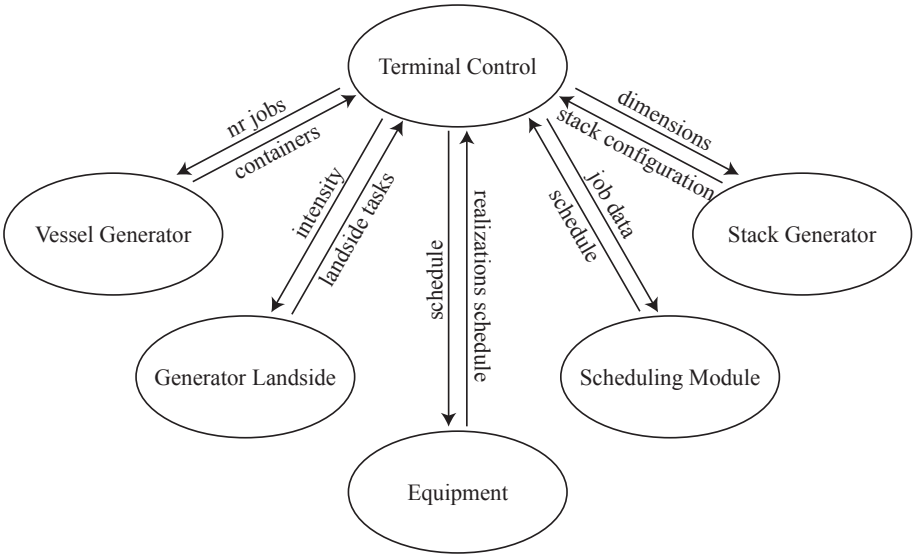

Figure 3.5: Relationships and information flows between simulation modules

3.4 Scheduling of handling equipment

Efficient scheduling of the handling equipment is crucial to have satisfactory performance of the container terminal. That is, the handling system of ASCs and AGVs should deliver the right container at the right time such that the QCs do not have to wait. In this thesis, we will consider the problem of determining the best order in which the ASCs and the AGVs should handle their containers such that the ship is loaded as fast as possible. This is typically an operational problem that has to be solved during every loading operation of a vessel. Throughout the thesis, we will make the following assumptions.

- The terminal layout, and the number of handling equipment is given. Determining the layout and the number of handling equipment are typically of a strategic and/or tactical nature and are beyond the scope of this research. For an overview of these problems, we refer to Chapter 2.
- We will consider only the loading operation of a vessel, since the unloading is less complicated (see Section 3.2).
- The stowage plan and the crane split is known. That is, for each QC we know which containers are handled by this QC and in which order.

- For each container, the location in the stack is known. Note that this already defines the ASC which will handle this container, since each stack lane has a dedicated ASC.
- All handling times are known, or at least estimates are known. In this thesis, we will consider two variants of the problem: deterministic and stochastic handling times.
- The traffic control of AGVs is not considered, since this problem appears on an even lower level in the decision hierarchy. Hence, we assume that there are no deadlock situations arising from poor traffic control. However, in Chapter 5 of this thesis, we will consider congestion, which results in deviations from the (expected) driving times of AGVs.
- Reshuffle moves are assumed to be carried out at the moment the container is retrieved from the stack. So, the handling time of a container includes the possible time necessary for reshuffles. In Chapter 7, we will shortly discuss how the models we present in this thesis can be extended in case reshuffles are allowed to be carried out earlier.

Our objective is to construct a detailed schedule for all the QCs, AGVs and ASCs, such that time at which the last container is loaded by any of the QCs is minimized. The schedule we construct determines the timing of containers loaded by the QCs. Moreover, the containers must be assigned to individual AGVs and the order and the timing of the containers assigned to the same AGV must be determined. Finally, for the ASCs, we already know which containers are handled by which ASC (since we know in which stack lane a container is stored). Thus, we have to determine the order and the timing of the containers on the individual ASCs.

3.5 Solution approach and motivation

To our knowledge, none of the existing literature explicitly deals with the integrated scheduling of the various types of handling equipment used at automated container terminals (see Section 2.4). A straightforward approach would be to divide the integrated problem into a number of subproblems, which then can be solved by using some optimization method known from literature, as discussed in Section 2.4. However, the fact that AGVs cannot load and unload containers themselves makes the schedules of the AGVs and the ASCs/QCs highly dependent on each other. Also the fixed loading order of the containers at the QCs is a complicating factor. In the

following subsections, we will give two examples which show why separate scheduling of handling equipment is not preferred.

3.5.1 Example 1: deadlocks

The following example shows that a non-integrated approach may lead to deadlocks, i.e., overall infeasible schedules. Obviously, the occurrence of deadlocks in automated container handling leads to a substantial loss of performance, since usually a part of the terminal has to be shut down in order to resolve the deadlock by manual intervention. Therefore, we feel that any optimization algorithm to be used, should guarantee feasibility of the schedules produced at all times.

Example
Consider a simplified situation in which we have a single QC, 2 AGVs and 3 ASCs. The QC has to handle containers 1, 2, 3 in this order. Each container is located in a different stack lane. Moreover the handling times of the containers on the ASCs 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 ASCs are ready to transfer the containers to the AGVs at times $t_1 = 3$, $t_2 = 2$, $t_3 = 1$. Suppose both the AGVs become available at time $t = 0$ and assume the driving times to the stack lanes are negligible. Now using for instance an AGV dispatching rule like First Come First Served (Egbelu and Tanchoco (1984); Van der Meer (2000)), which assigns an idle vehicle to the load that is available longest, leads to the assignment of containers 2 and 3 to the two AGVs. This is because containers 2 and 3 are available for transport earliest (at time $t = 2$ and $t = 1$, respectively). However, after this assignment, there is no empty AGV left to transport container 1, which is required first at the QC. As a result, both AGVs cannot be released from their container and also the ASC that is retrieving container 1 cannot be released. This results in a deadlock situation which is illustrated in Figure 3.6.

3.5.2 Example 2: low performance

One might argue that while solving the subproblems (for instance the scheduling of an ASC), the order of the containers at the QC should be taken into account. In this way, deadlocks as illustrated in the previous example might be avoided. However, this may lead to schedules which are far from optimal, as the following example illustrates.

Example
Consider again a single QC with containers $1, 2, 3, 4$ which are to be handled in this

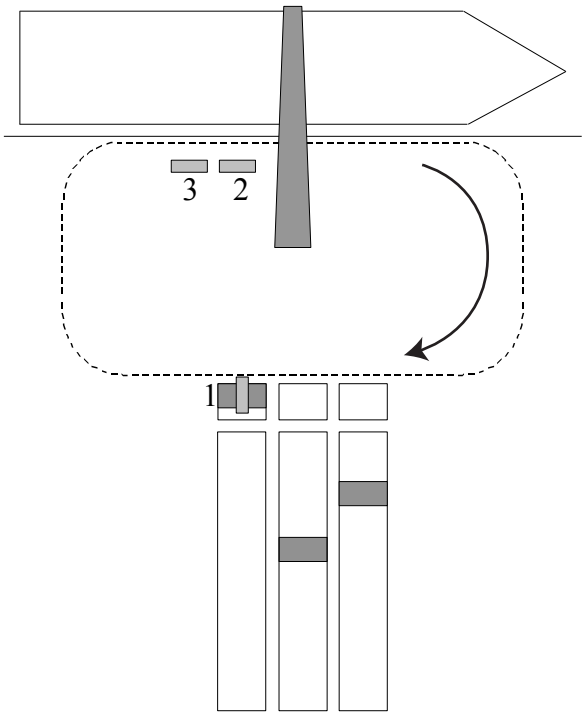

Figure 3.6: Example 1: a deadlock situation

order and which handling times on the QC are negligible. Moreover, assume there is only a single ASC that has to handle all four containers. The handling times of the containers on the ASC are given by $p_1^{asc} = 1$, $p_2^{asc} = 1$, $p_3^{asc} = 1$, $p_4^{asc} = 9$. Consider two AGVs and let the driving times from the stack to the QC (p_i^{agv}) and vice versa (d_i^{agv}) be the same for each container, that is $p_i^{agv} = 5$, $d_i^{agv} = 5$, $\forall i := 1 \dots 4$. Suppose the objective is to minimize the makespan of the schedule. Obviously, a schedule for the AGVs that completely respects the order of the containers at the QC is the following: AGV_1: containers 1 and 3, AGV_2: containers 2 and 4. Moreover, taking the order of the containers at the QC into account, this leads to the order $1, 2, 3, 4$ on the ASC. This results in a makespan of 25, where reversing containers 3 and 4 on the ASC results in a makespan of only 17 (an improvement of more than 30 percent). The Gantt charts in figure 3.7 illustrate both solutions. Note the idle time between operations. For instance, in the lower chart, AGV_2 has to wait with transporting container 4, since container 3 has to precede on the QC.

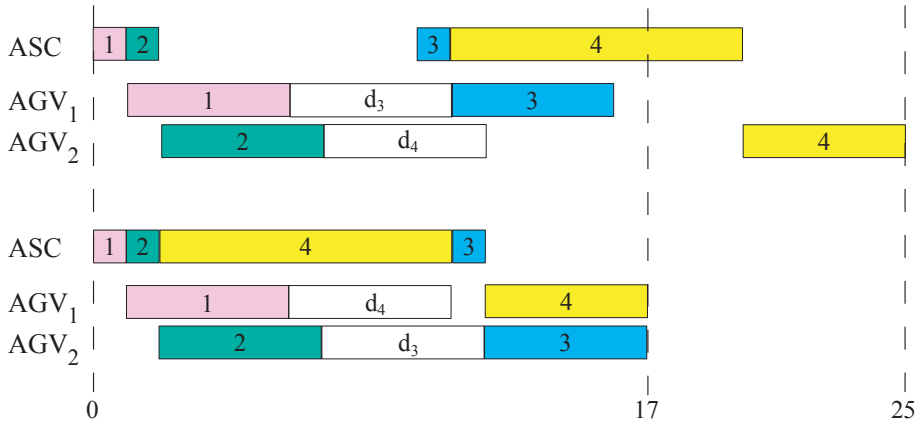

Figure 3.7: Example 2: Gantt charts of solution based on non-integrated scheduling (upper) and optimal solution (lower)

3.5.3 Solution approach

Examples 1 and 2 show that solving subproblems, instead of the integrated scheduling problem, does lead to a poor performance or even deadlock situations. Although Example 1 may seem to be somewhat artificial, it illustrates perfectly the deadlocks we experienced after implementing simple decision rules into our detailed simulation model. Hence, we propose an integrated approach for the scheduling of ASCs, AGVs and QCs. However, as the result of Theorem 4.1 in Chapter 4 will show, there is not much hope for finding a polynomial time algorithm to solve this scheduling problem. In this thesis, we will therefore derive both exact enumeration algorithms in order to find optimal solutions, as well as heuristic procedures. In Chapter 4, we will discuss a Branch & Bound algorithm, that can be used to construct schedules for the handling equipment given the existing terminal layouts. Based on this Branch & Bound algorithm, we also develop a heuristic Beam Search algorithm. In Chapter 5, we apply this Beam Search algorithm in a dynamic context and we also introduce several dispatching rules for which we *can* guarantee feasible solutions. Finally, in Chapter 6, we consider a terminal with a general layout. For this case, we model the problem as a Mixed Integer Program (MIP) and give several classes of valid inequalities that help in reducing the computational effort to solve the model.

Chapter 4

Integrated scheduling of handling equipment: the static case

In this chapter we consider the problem of integrated scheduling of various types of handling equipment at an automated container terminal. We assume a deterministic scenario, that is, we assume that all handling times are known in advance. Moreover, we will consider layouts in which all AGVs pass a common point after unloading at the QCs. For such a layout, we can represent the schedules of both the ASCs and AGVs in a concise way. Based on this representation, we will develop a Branch & Bound algorithm that uses various combinatorial lower bounds. Moreover, we also discuss a Beam Search heuristic that is based on this Branch & Bound algorithm.

This chapter is organized as follows. In the next section, we will shortly discuss the specific problem setting we consider. In Section 4.2, we will model the problem, discuss its complexity and derive some results that will be used in a Branch & Bound algorithm that is presented in Section 4.3. A Beam Search heuristic, based on this Branch & Bound algorithm, is discussed in Section 4.4. Computational results are given in Section 4.5, followed by conclusions in Section 4.6.

4.1 Problem definition

In this chapter, we consider the problem of scheduling ASCs, AGVs and QCs in an integrated way, as discussed in Chapter 3. In particular, we consider a deterministic

situation. Hence, we assume that all handling times of the containers on the various equipment is known.

Recall the overview of ECT's automated container terminal as was shown in Figure 3.1. In Figure 4.1, we give a more schematic layout of this terminal.

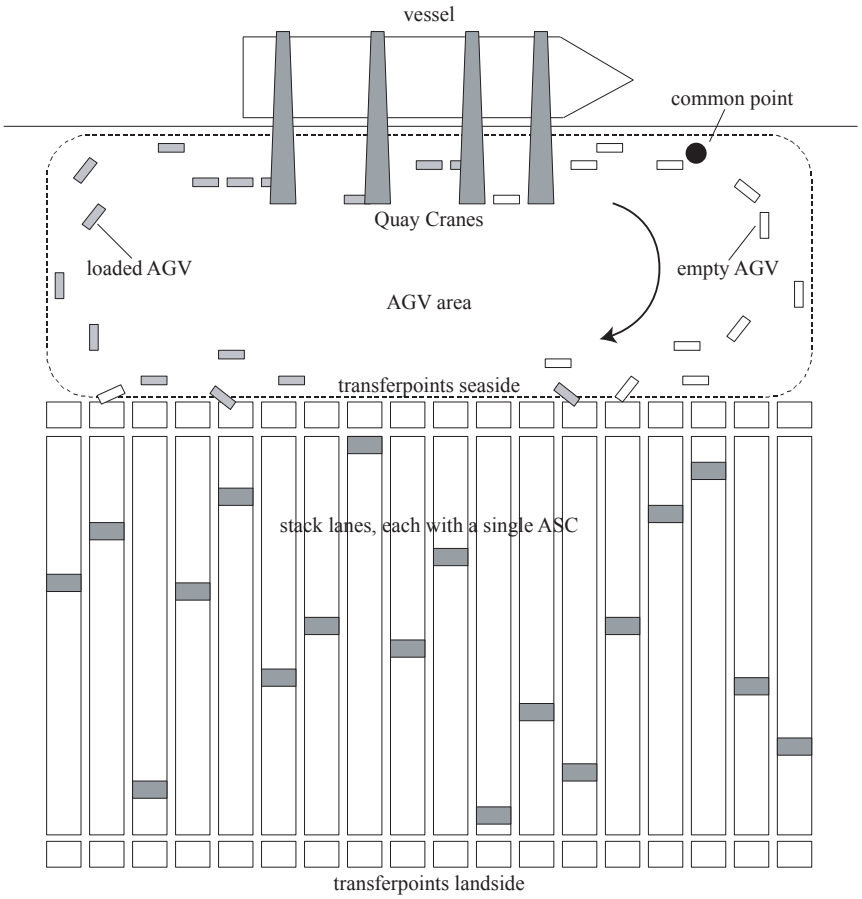

Figure 4.1: Schematic layout of an automated container terminal

From Figure 4.1, we can see that the AGVs are driving in a clock-wise loop. The transparent AGVs are empty, the shaded AGVs are loaded. Note that after the AGVs are unloaded by one of the QCs, they all pass a common point on their way to the next container to be picked up at a stack lane. This characteristic of the layout allows us to represent the schedules of both the ASCs and the AGVs in a concise

way, as we will see later in this chapter.

As already mentioned before in Chapter 3, all existing literature considers the scheduling of only a single type of equipment (see Section 2.4). 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 with its next container 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, as was shown in Chapter 3. 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.

4.2 Modeling and complexity

In this section we will model the scheduling problem of ASCs, AGVs and QCs for the case in which we have a layout as shown in Figure 4.1. 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 4.3). However, we will first discuss some modeling issues.

The loading of a single container corresponds to three tasks, which are carried out by different types of equipment: the ASC, the AGV and finally, the QC. We will refer to these tasks as the ASC/AGV/QC task. 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 QCs, ASCs and AGVs, respectively. For the three tasks of which the handling of a container consists of, we introduce the parameters $p_i^{asc}, p_i^{agv}, p_i^{qc}$ to represent the handling times of container i on the ASC, AGV and QC, respectively. Moreover, for each AGV task we introduce two additional drive times d_{i1}^{agv} and d_{i2}^{agv}. These parameters represent the drive time from the common point to the proper stack lane (where the container is stored) and the drive time from the QC (where the AGV is unloaded) to the common point, respectively.

The handling 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 handling and the drive 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 hold of the ship to which the container was assigned in the stowage plan. We assume that all handling and driving times are deterministic and that preemption is not allowed. For instance, once an AGV has started to transport a container, this can not be modified anymore, even if the AGV is still on its way to the stack to pick up the container.

4.2.1 Blocking constraints

An important characteristic of the AGVs 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 its next task 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 its next task until the QC has lifted the container off the AGV. So, the QC blocks the AGV.

4.2.2 Time-lags

Consider the loading sequence of a single QC, which follows from the stowage plan and the crane split (see Section 3.2). Consider two containers i and j from this sequence, which are loaded by the QC directly after each other. Then 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 QCs not explicitly, but to represent them 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 directly 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 containers.

4.2.3 One-to-one correspondence QC - stack

Obviously, there is a one-to-one correspondence between the containers to be loaded by the QCs and the containers located in the stack. 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 contents of the subsets are known in advance.

4.2.4 Complexity

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

Theorem 4.1 *The integrated scheduling problem of $|Q|$ QCs, $|M|$ AGVs and $|S|$ ASCs as described in this section is NP-hard.*

Proof: The problem of minimizing the makespan 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_{ij} 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 there exists a feasible schedule with makespan no more than 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 lane and their ASC handling times are equal to $\epsilon < 1$.
3. There is a one-to-one correspondence between chains and QCs. The order in which a QC handles its containers is exactly the order that is defined by the corresponding chain. If container i is the immediate predecessor of container j in one of these chains, then the handling 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 QCs 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 handling 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 handled 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 AGVs. Add to the constructed instance above an arbitrary number of AGVs and the same number of containers. All these containers are located in a second stack lane and all need to be transported to an additional QC that is located "far away" from the second stack and from the other QCs. The handling 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 to $\bar{C}_{max}$ if all additional AGVs leave almost immediately to pick up one of the additional containers. We leave it to the reader to work out the details. □

4.2.5 Dominant schedules

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 the containers are assigned to the AGVs. Hence, enumerating over all possible orders will provide us with the optimal solution. First, we observe the following:

Observation 4.2.1 *We may restrict ourselves to schedules in which waiting times of the AGVs only occur due to the fact that the proper ASC or QC is not ready to load or unload a container.*

Now recall the common point, as illustrated in Figure 4.1, which every 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 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.

Next, we will show that we may restrict ourselves to schedules for which the assignment order is consistent with the orders in which the ASCs handle the containers.

Definition 4.2 *Define the assignment order π as the order in which the containers are assigned to the (empty) AGVs as they pass the common point. Moreover, define a suborder π_s as a subset of π such that if i is ordered before j in π_s, then i is ordered before j in π, for all $i, j \in \pi_s$.*

Theorem 4.3 *Consider an optimal schedule. Let π_s denote the order in which ASC s handles its containers. Then there exists an optimal assignment order π, 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 containers i, j such that i precedes j in π_s and j precedes i in π. Next, we will show that reversing the containers i and j in π, will give a schedule with the same makespan.

Let c_i and c_j denote the completion times of the ASC tasks related to containers i and j. 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 AGV assigned to container j arrived at the ASC not later than the AGV assigned to container i. Hence, the AGV assigned to container j is also available at time c_i. Now consider the same assignment order, except that containers i and j swap positions. It is obvious that both AGVs are available again at the ASC at time c_i. Hence, the completion times of tasks 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 containers i, j for which the orders π and π_s do not correspond, we get the required result. □

Theorem 4.3 states that in search for the optimal schedule (with the smallest makespan), we may restrict ourselves to the schedules in which the order in which the containers are assigned to the AGVs is consistent with the order in which the

handling time on:	container 1	2	3	4	5
ASC_1	100		30		50
ASC_2		25		70	
AGV (QC-stack)	75	50	75	50	75
AGV (stack-QC)	50	75	50	75	50
QC_1	50	60	70		
QC_2				75	40

Table 4.1: Handling and driving time of containers on equipment

containers are scheduled on the ASCs. Hence, we have that the assignment order π completely determines the order of the containers both on the AGVs and the ASCs. Moreover, in combination with Observation 4.2.1, the assignment order π specifies the complete schedule, which is illustrated by the following example.

Example
Consider a terminal with 2 QCs, 2 ASCs and 3 AGVs. Suppose there are 5 containers to be loaded: containers 1,2 and 3 in this order by QC_1 and containers 4 and 5 in this order by QC_2. Furthermore, containers 1, 3 and 5 are located in the stack lane of ASC_1, containers 2 and 4 in the stack lane of ASC_2. All AGVs are initially located at the common point. For convenience, assume the distance between the QCs is negligible (located at the same position) and that the common point is located at the position of the QCs. Table 4.1 gives the handling and driving times of the containers on the various pieces of equipment. Now suppose we have an assignment order π which is as follows:

$$\pi = \{1, 4, 2, 3, 5\} \tag{4.1}$$

Given Theorem 4.3, we then have the suborders for the ASCs:

$$\pi_1 = \{1, 3, 5\} \tag{4.2}$$

and

$$\pi_2 = \{4, 2\} \tag{4.3}$$

Executing this schedule will initially assign container 1 to AGV_1, container 4 to AGV_2 and container 2 to AGV_3. As a result of Observation 4.2.1 we have that a piece

	container				
	1	2	3	4	5
ASC_1	[0, 100]				
ASC_2		[70, 95]		[0, 70]	
AGV_1	[0, 150]				
AGV_2				[0, 145]	
AGV_3		[0, 200]			
QC_1	[150, 200]	[200, 260]			
QC_1				[145, 220]	

Table 4.2: Example: partial schedule resulting from assignment order

	container				
	1	2	3	4	5
ASC_1	[0, 100]		[100, 220]		[220, 270]
ASC_2		[70, 95]		[0, 70]	
AGV_1	[0, 150]				[150, 320]
AGV_2			[145, 270]	[0, 145]	
AGV_3		[0, 200]			
QC_1	[150, 200]	[200, 260]	[270, 340]		
QC_1				[145, 220]	[320, 360]

Table 4.3: Example: complete schedule resulting from assignment order

of equipment starts with its next container immediately after the previous container has been handled. This yields the partial schedule for the first three containers in the assignment order (1, 4, 2), which is illustrated in Table 4.2.

An entry in Table 4.2 gives the time interval in which a certain piece of equipment is handling a certain container. Note that this may include waiting time. For instance, AGV_3 handles container 2 in the interval [0, 200]. First, the AGV drives from the common point to the proper stack lane, i.e., ASC_2. However, at the stack lane the AGV has to wait, since according to the schedule (π_2), ASC_2 handles container 4 first. As a result, AGV_3 can only start driving to QC_1 at time $t = 95$ and arrives at QC_1 at time $t = 170$. However, AGV_3 can not be unloaded until time $t = 200$, the time at which QC_1 is ready with loading container 1 (which should precede container 2 on the QC). In Table 4.3, we illustrate the complete schedule for all containers.

Now consider for instance AGV_2 in Table 4.3, which is the first of the three AGVs that becomes idle again. According to the assignment order, this AGV is assigned to container 3. Since AGV_2 becomes idle at $t = 145$, it arrives at ASC_1 at time $t = 220$. Note from Table 4.3 that ASC_1 already started handling container 3 at time $t = 100$. The earliest time at which ASC_1 could finish with container 3 is at time $t = 130$, but it has to wait for the AGV until time $t = 220$. After container 3 has been loaded, AGV_2 can advance to QC_1 at which the AGV arrives at time $t = 270$. Then it is immediately unloaded by QC_1, since QC_1 has been idle since time $t = 260$.

The example above illustrates how the assignment order π, although it only seems to consider the AGV assignment, completely determines the remaining part of the schedule. Moreover, Theorem 4.3 states that we can restrict ourselves to these types of schedules without loss of optimality. Therefore, by enumerating over all possible assignment orders π, we can find the optimal schedule. In order to speed up this enumeration, we will develop a Branch & Bound algorithm which is discussed in Section 4.3.

4.2.6 Feasible partial schedules

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

Definition 4.4 *Let π' be a partial assignment order of the first k containers. 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 AGVs the assigned containers 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 can not be extended to a complete feasible schedule since no AGV has a finite finish time. However, it will be shown in Subsection 4.3.4 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 containers are handled.

4.3 A Branch & Bound algorithm

Because of Theorem 4.3, 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.

4.3.1 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 containers. If this partial assignment is feasible (see Definition 4.4), we can determine the finish times of the QCs and the ASCs and of at least one AGV.

The branching rule is then simply as follows. For each subproblem with the first k containers fixed, we introduce branches by determining which container 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 containers 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.

4.3.2 Search strategy and global lower bound

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 breadth-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 less of the value of this best 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 node. 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 in successive runs. 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 4.3.5, the Branch & Bound algorithm is stopped if the number of generated 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.

4.3.3 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.

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 containers. From the order of these containers, we obtain for each QC the last container that it handles and the completion time of this container (see also Section 4.2). Denote by i_q the last container handled by QC q and let f_q be the corresponding completion time on the QC of this container. Moreover, let $j \succ i$ represent the precedence relations on the QC, that is, container j succeeds container i. Then we can calculate for each QC a lower bound on the makespan C_{max} as follows:

$$\max_{q=1,2,\dots,|Q|} \{f_q + \sum_{l \succ i_q} p_l^{qc}\} \tag{4.4}$$

One Machine Scheduling lower bound

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

For the remaining containers 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 related tasks on the AGVs are omitted. As a consequence, the remaining containers will be handled on the ASC with no idle time.

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

Theorem 4.5 *Let f_s denote the completion time of the last scheduled container on ASC s. Moreover, w.l.o.g. assume that the remaining containers $1 \dots n_s$ to be scheduled on ASC s are numbered in order of non-increasing tail, where the tail of*

container i is defined as:

$$t_i^{asc} := p_i^{agv} + \sum_{l \succeq i} p_l^{qc} \tag{4.5}$$

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

$$\max_{s=1,2,\dots,|S|} \max_{i=1,2,\dots,n_s} \{f_s + \sum_{l=1}^{i} p_l^{asc} + t_i^{asc}\} \tag{4.6}$$

Proof: Equation (4.6) follows from the fact that, given unlimited AGV capacity, it is optimal to schedule the containers on the ASC in order of non-increasing tail. To see that this is indeed optimal, suppose that there is a pair of containers 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 handling time on the ASC of all the containers in $\{1, 2, \dots, n_s\}$ that are scheduled before j in this optimal solution. Since j is scheduled immediately before i and $t_i^{asc} > t_j^{asc}$, we have

$$f_s + PP_j + p_j^{asc} + p_i^{asc} + t_i^{asc} > f_s + PP_j + p_j^{asc} + t_j^{asc} \tag{4.7}$$

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 containers i and j does not increase the makespan, since

$$\begin{aligned} f_s + PP_j + p_j^{asc} + p_i^{asc} + t_i^{asc} > \\ \max\{f_s + PP_j + p_i^{asc} + t_i^{asc}, f_s + PP_j + p_i^{asc} + p_j^{asc} + t_j^{asc}\} \end{aligned} \tag{4.8}$$

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

Note that the bound (4.6) is calculated for all unassigned containers, since we consider all ASCs and for each ASC we consider all containers to be scheduled on this ASC.

Parallel Machine Scheduling lower bound

The following lower bound is based on considering the AGVs 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 stack lane (ASC) and the driving time from the QC to the common point to the handling time on the AGV. So, for each container we redefine the handling time on the AGV as:

$$\tilde{p}_i := d_{i1}^{agv} + p_i^{agv} + d_{i2}^{agv} \tag{4.9}$$

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

$$\tilde{p}_i := \min_{j=1,2,\dots,n} \{d_{j1}^{agv} + p_j^{agv} + d_{j2}^{agv}\} \tag{4.10}$$

Similar to the tail defined in the previous subsection by equation (4.5), we define for each container a tail t_i^{agv} as follows:

$$t_i^{agv} = \sum_{l \succeq i} p_l^{qc} \tag{4.11}$$

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

$$\min \max_{1,2,\dots,n} \{c_i + t_i^{agv}\} \tag{4.12}$$

where c_i is the completion time of container i on its AGV. Since all AGV handling times are equal for all containers, it is easily seen that the containers should be handled in order of non-increasing tail in order to obtain the minimum in (4.12). Note that this means that we implicitly respect the precedence constraints among the containers belonging to the same QC. This is because for every pair of containers i, j such that i precedes j on the QC, we have that $t_i^{asc} > t_j^{asc}$. However, we do not necessarily respect the time-lags among the related AGV tasks. That is, for two consecutive containers i, j on a QC, we do not necessarily have that AGV task j is completed at least p_i^{qc} time later than AGV task i is completed.

In general, we have a partial feasible assignment order (Definition 4.4) and a corresponding partial schedule when the above lower bound is calculated in a node of the tree. The idea is to handle the remaining (unassigned) containers on the AGVs in the order indicated above as soon as the AGVs become available. Recall, however, that there may be AGVs for which the completion time of their current task can not be determined yet. Therefore, we derive a lower bound on the earliest possible time that these AGVs 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 Quay Crane lower bound as discussed earlier. Let i be the container handled by the AGV and let j be the last container 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 container i by summing

up all handling times on the QC of containers $j+1, \ldots, i-1$. 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 One Machine Scheduling lower bound as discussed earlier. Let i be again the container handled by the AGV and let j be the last container handled by the QC. Now define H_s to be the set of containers 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 container i to be released as soon as possible, the ASC should first handle all the containers in set H_s, in the same order in which the containers 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 container i is assigned:

$$f_i = \max_{s \in S} \max_{h \in H_s} \left\{ \sum_{l \in H_s:\ l \preceq h} p_k^{asc} + \sum_{l \succeq h}^{l \prec i} p_l^{qc} \right\} \tag{4.13}$$

4.3.4 Upper bound heuristic

In each node of the branching tree, the partial assignment of the first k containers is completed by applying the following heuristic. The unassigned containers are sorted in order of non-increasing tail t_i^{agv}, as given by (4.11) and are added in this order to the partial assignment. As a result, we obtain a complete assignment. Now suppose that the partial assignment is feasible, as defined in Definition 4.4. We will show below that the complete assignment obtained by the heuristic is feasible in the sense that all containers will be completed. Clearly, the objective value of any feasible solution is an upper bound on the makespan.

Theorem 4.6 *Consider any partial assignment of the first $k < n$ containers which is feasible according to Definition 4.4. Completing the assignment by adding the unassigned containers 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 containers in order of non-increasing tails, it is sufficient to show that adding the first container 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 container, again at least one AGV will have a finite finish time.

First, observe the following. By definition of the tails t_i^{agv} (4.11), the following property is satisfied. For each pair of containers i, j such that container i precedes container j at a QC, we have:

$$t_i^{agv} > t_j^{agv} \tag{4.14}$$

Hence, the container to be added to the partial assignment is a container 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 AGVs has a finite finish time. Hence, the added container is handled by an AGV that originally had finite finish time. This AGV picks up the container (which is possible since all ASCs have finite finish time in the original (feasible) partial schedule) and transports it to the QC, which handles it immediately. As a result, the AGV has a finite finish time again. □

4.3.5 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.

4.4 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.

4.4.1 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 containers 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 4.2 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.

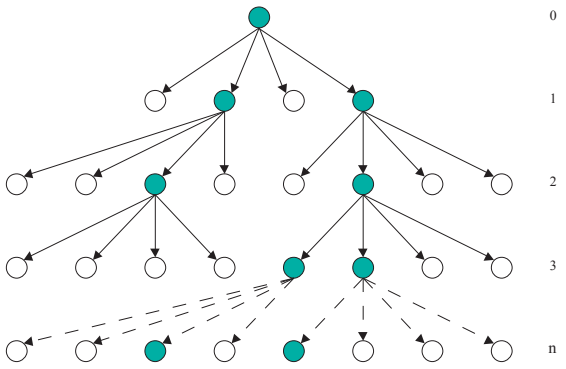

Figure 4.2: Beam Search algorithm with beam width equal to 2

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 and Morton (1988), Sabuncuoglu and Karabuk (1998), Sabuncuoglu and Bayiz (1999), and Kim and Kim (1999*c*).

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 selection 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 4.3. 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.

4.4.2 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 which were discussed in Subsection 4.3.3. Since at level k, all nodes represent an assignment schedule for the first k containers, the lower bound gives an indication of how "efficient" the first k containers 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 containers, as discussed in Subsection 4.3.4. Like in the Branch & Bound algorithm, we keep track of the best solution found so far. Obviously, whenever the lower bound of a node is higher than 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.

4.4.3 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 container (the node at the first level of the tree), and completing the schedule by selecting at each level the most promising container. However, this means that the nature of the search tree is omitted, since the search tree allows us to select a somewhat less promising container 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.

4.4.4 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 containers are sorted in order of non-increasing tail t_i^{agv} (4.11). Now consider a node in the upper part of the tree, that is, a node for which only a small number of containers have been assigned yet. It is unlikely that scheduling the container with the smallest tail next, will give us a good solution. This is because the container with the smallest tail is a container which is scheduled last on some QC. So, if this container 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 each of the *fw* unassigned containers 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 4.5, 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.

4.5 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 containers. The medium and large size problems contain more than 75 and 160 containers, respectively. All instances were obtained from the detailed simulation model, which was developed in close cooperation with ECT, the operator of the automated terminals in Rotterdam (see Section 3.3 for more details). The simulation model allows for experimenting with different layouts of the container terminal, different number and speed of equipment, etcetera. Besides, for the same set of containers, the same loading plan and the same positions of the containers in the stack, we can construct different instances. This is done by varying the number of AGVs used, the driving speed of the AGVs and/or the handling speed of the ASCs. For the QCs, the handling speed was not varied, since this is still a manual operation and thus can not be carried out faster. Note however, that for each container the handling time on the QC differs, depending on the position of the container in the hold (see also Subsection 3.3.4). So, in Table 4.4 and Table 4.6, each line represents a different instance of the problem.

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

4.5.1 Results for the Branch & Bound algorithm

Table 4.4 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 (QCs, ASCs and AGVs). 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 4.4, 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

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 4.4: Computational results for the Branch & Bound algorithm

5 percent of the optimum. Keep 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 4.4, the value of the solution found represents the makespan 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.

4.5.2 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 4.3 shows the results of the Beam Search algorithm for various beam and filter widths[1], based on computational data obtained from solving ten large instances (more than 160 containers). 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 4.3 we may conclude that the parameter settings of the beam and filter width do 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 4.3 also shows that taking a larger beam width does not necessarily lead to a better solution. As pointed out before in Subsection 4.4.3, 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.

[1]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

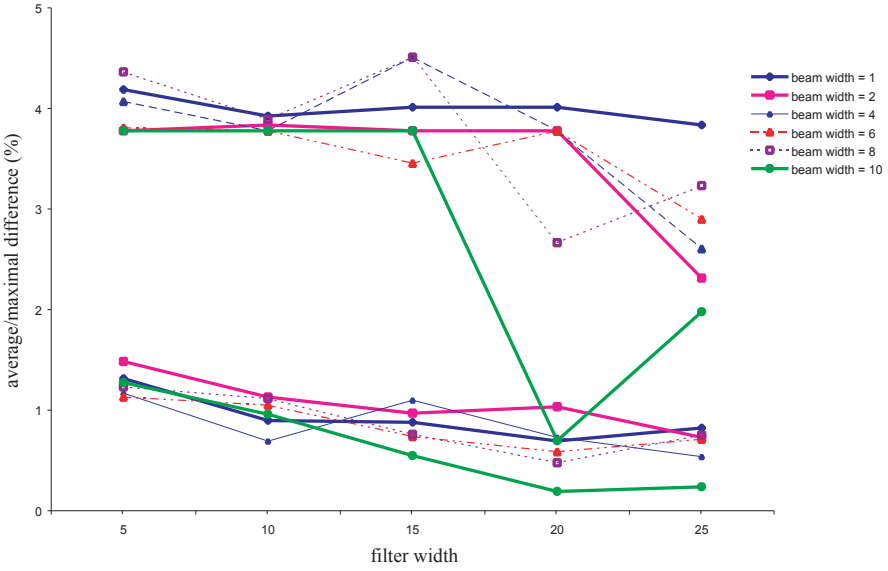

Figure 4.3: Average and maximal difference between beam and filter width combination and the best beam and filter width combination

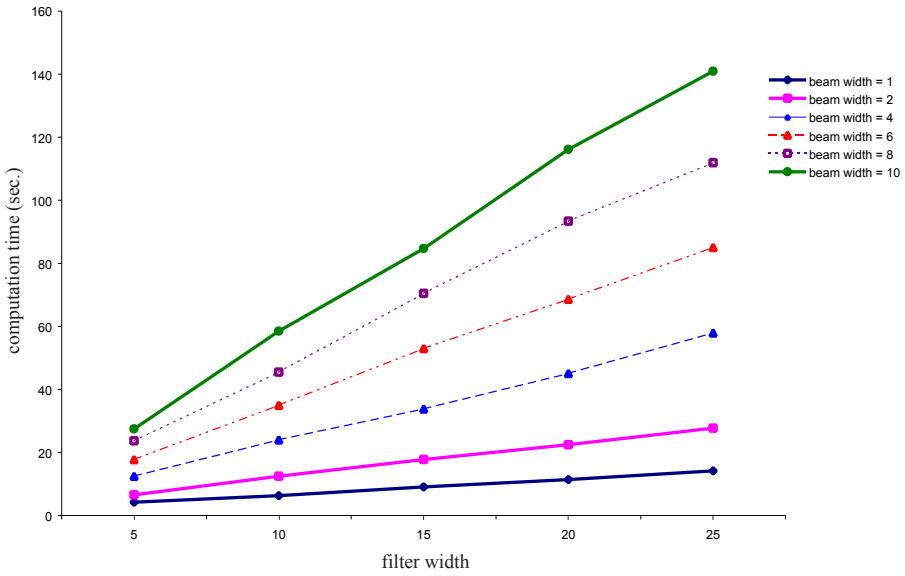

Figure 4.4: Effects of increasing filter width, for fixed beam width, on computation time

Figure 4.4 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 $\mathcal{O}(bw \cdot fw \cdot n)$ in case of applying a filter, gives a good estimate of the relative computation times under different beam and filter widths.

Combining Figure 4.3 and Figure 4.4, 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 are sometimes significant. In Table 4.5 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 4.5, for a given time of, for instance, 30 seconds, the combination $bw = 4$, $fw = 10$ yields the best results. For this combination, the

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

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.

4.5.3 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 4.5 it follows that for a maximal computation time of 30 seconds, this combination gives the best results.

In Table 4.6, the first four columns specify again the size of the problem. Next, we repeat partially the results of Table 4.4 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 4.6 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 4.4).

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 4.6: Computational results for the Beam Search algorithm with $bw = 4$ and $fw = 10$ versus results for the Branch & Bound algorithm

4.6 Conclusions and further research

In this chapter, 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 AGVs pass a common point after they are unloaded at the QCs. 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 containers to the AGVs is consistent with the order of the containers on each of the individual ASCs. Hence, the assignment order of the containers to the AGVs 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 4.5 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 handling times (see also Chapter 5).

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.

Chapter 5

Dynamic scheduling of handling equipment

In the previous chapter, we considered the integrated scheduling of various types of handling equipment at an automated container terminal in a static context, i.e., we assumed that all the information about the containers to be handled is known beforehand.

In this chapter, however, we consider this scheduling problem within 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. Within this dynamic context, we investigate the performance of the Beam Search algorithms presented in Chapter 4. We test the algorithm using different planning horizons and different frequencies of updating the schedules. Moreover, we investigate the performance under stochastic handling times.

As an alternative to the optimization based Beam Search heuristic, 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 chapter is organized as follows. Next, in Section 5.1, we will discuss how the Beam Search algorithm presented in Chapter 4 can be adjusted for a dynamic setting. In Section 5.2, we will report on our computational experiments with this approach both in the deterministic and stochastic case. Next, in Section 5.3, we will discuss several dispatching rules known from the literature and we will show that adjustments are necessary in order to guarantee feasible schedules.

Computational results of these dispatching rules will be discussed in Section 5.4. Finally, in Section 5.5 we will make some concluding remarks.

5.1 Beam Search in a dynamic setting

The Beam Search algorithm of Chapter 4 is based on (partially) enumerating all the assignment orders π. The assignment order π represents the order in which the containers are assigned to the AGVs that become idle. The order of the containers on the ASCs is then kept in accordance with this assignment order π. It was shown in Theorem 4.3 that there is always an optimal schedule in which this is the case.

Next, we will discuss how the Beam Search algorithm can be applied in a dynamic setting. In a dynamic setting, 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 this will happen when we apply 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 given a 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 following example illustrates this.

Example

Given are 4 containers to be handled, 1 ASC and 2 AGVs and assume the QC handling times to be negligible. The top Gantt chart in Figure 5.1 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 driving times back to the stack lane (d_3 and d_4) after the AGVs have completed the handling of their first container. The bottom Gantt chart in Figure 5.1 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 AGVs.

Now suppose that containers are rescheduled, based on updated information about the handling time of the containers. Given the part of the schedule that

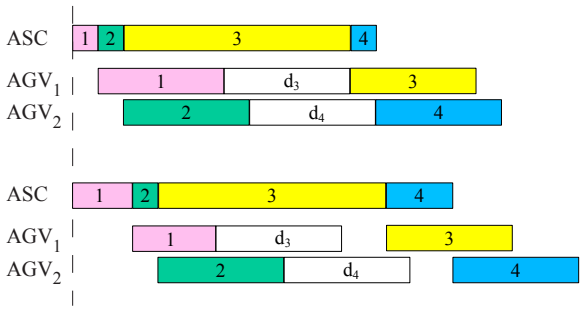

Figure 5.1: Gantt charts

has already been executed, it is not obvious that the Beam Search algorithm may find a feasible schedule. This is because the Beam Search does not exploit the search tree completely. Hence, the node representing the (obviously) feasible solution (as shown in Figure 5.1) 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 containers to the AGVs (and the ASCs), as defined in Theorem 4.3. 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. This is illustrated in the example in Figure 5.2. Let container 4 be the container with the largest position in the assignment order that is currently handled by some piece of equipment. Then we may reschedule only the containers 5, 6 and 7.

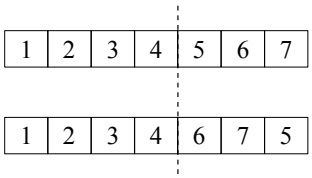

Figure 5.2: Rescheduling containers

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". Note that by definition, the Beam Search investigates only a part of the search tree. 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 QCs, 2 AGVs and 2 ASCs. The QCs load containers 1,2 and 3,4 respectively. ASC_1 has to handle containers 1 and 3, ASC_2 handles containers 2 and 4. Suppose we have the initial schedule, represented by the assignment order $\pi = \{1, 3, 4, 2\}$. Whenever we execute this schedule, ASC_1 starts with container 1, ASC_2 starts with container 4. Suppose we reschedule after container 1 has been handled by the ASC. Moreover, suppose that the tails t_i^{agv} (defined by equation (4.11)) of the containers 2, 3 and 4 are equal to 50, 100 and 49, respectively. In the Beam Search algorithm, the containers are initially sorted in order of non-increasing tails (see Subsection 4.4.4). Hence, initially, we have the assignment order $\pi = \{3, 2, 4\}$. Now suppose we run the algorithm with, for example, filter width $fw = 2$ and beam width $bw = 1$. We then get the following search tree, which is illustrated in Figure 5.3.

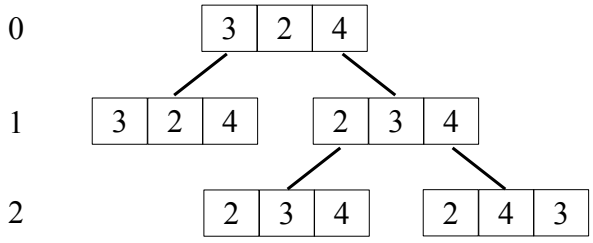

Figure 5.3: Search tree Beam Search, $bw = 1$, $fw = 2$

At level 0, we have the containers sorted according to their tail t_i^{agv}. With a filter width 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 right 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. So, for this particular example, the assignment order up till container 4 has to be taken fixed.

5.2 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. More 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.) In combination with (a.), we investigate 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 QCs, since this is still a manual operation. Moreover, there is uncertainty in the driving times of the AGVs, since some congestion may appear near crossings of AGV tracks. The uncertainty in the handling times of the ASCs is limited, since an ASC operates fully automated and there is no interference with other ASCs. We will therefore consider only uncertainty in the handling times of containers on the QCs and the AGVs. 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 is 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.

The overall performance is measured in terms of the in-port time of the vessel. The minimization of the in-port time is very important because of the high operating costs of the vessels and the tight schedules they have to meet. Since we consider only

the loading of a vessel in this thesis, this is reflected by the minimization of the time at which the last container is loaded, i.e., the minimization of the makespan of the schedule.

In order to investigate the performance of various algorithms for the scheduling of terminal equipment, we implement the algorithms in the detailed simulation model that was developed and which was discussed in Section 3.3. In the experiments that were used to test the performance of the Beam Search algorithm, approximately 1000 containers were generated, to be loaded by 4 QCs. Again, as in the computational experiments of Chapter 4, we test the algorithm for different parameter settings of the simulation model, like the number of AGVs, the speed of the AGV's, etcetera. In order to guarantee reliable outcomes, over 150 loading operations were simulated. In Table 5.1, we show the relative performance under various settings for both the planning horizon and the frequency of rescheduling, in a complete 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 to be loaded at once at the beginning of the operation, and keeping this schedule fixed. In the remainder of this chapter, we will refer to this as the "static" version of the Beam Search algorithm. The numbers given in Table 5.1 represent the average 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. Computation times for the various settings of the planning horizon and the rescheduling interval varied from less than 1 minute to 5 minutes for constructing the initial schedule, and at most 2 minutes for rescheduling. Note that whenever we reschedule, a part of the schedule is fixed (see Subsection 5.1) and thus, computation times are shorter. Scheduling all 1000 containers at once (the static version of the algorithm), led to computation times which were around half an hour. All computations were done on a Pentium PC 400 MHz.

The results from Table 5.1 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 provide us a lower bound on the time required to load the vessel and hence, would perform best for every problem instance.) Moreover, we may conclude from Table 5.1 that the length of the planning horizon is more crucial to achieve satisfactory performance than the frequency of rescheduling. Taking a short planning horizon of

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 5.1: Performance (in percentage deviation from makespan of static schedule) under different planning horizons / rescheduling intervals

about 20 or 30 containers per QC, which represents about half an hour of workload, yields a relatively poor 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.

The results from Table 5.1 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 poor schedules, compared to the schedules obtained by considering all containers at once. For the remainder of the instances, the schedules result in an almost similar performance. 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 version performed in the best case about 5 percent better than the static version (note 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 QCs and the AGVs 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 5.2 and Table 5.3 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 5.2: Performance (in percentage deviation from makespan of static schedule) dynamic scheduling vs. static scheduling, low stochastic

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 5.3: Performance (in percentage deviation from makespan of static schedule) dynamic scheduling vs. static scheduling, high stochastic

From the results of Table 5.2 and Table 5.3, we may again conclude that the length of the planning horizon is the most important. However, 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 Table 5.2 and Table 5.3 we may therefore conclude that such a schedule is quite robust. This is

also illustrated in Table 5.4, which gives the difference in performance of the static schedules whenever they are evaluated in a more stochastic scenario. As we can see from the table, the influence of the stochasticity of the handling times of the containers on the QCs and the AGVs on the overall performance is limited.

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

Table 5.4: Impact of stochastic handling times on overall performance of static schedule

5.3 Dispatching rules

The advantages of dispatching rules are obvious. By definition, these rules do not use complicated underlying mathematical models to calculate the "best" assignment. Hence, they are easy to implement and therefore often used by practitioners. Moreover, these rules require only little information about the system. For instance, when 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 not only compare the performance of various dispatching rules with each other, we will also compare their performance with the more complicated Beam Search algorithm.

5.3.1 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 and Tanchoco (1984), who were among the first to investigate the performance of various dispatching rules for AGVs 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 and 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 and Tanchoco (1984)). Assigns an idle AGV to a randomly chosen load.

4. Fixed Assignment of AGVs to QCs. 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 throughput times
- Minimization of queue lengths
- Minimization of (empty) driving distance
- Balanced workload

It is obvious that the behavior of some dispatching rules favors one 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 AGVs. However, 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 ASCs has to be determined, which must be coordinated with the AGV dispatching. In Subsection 5.3.3, we will discuss how the dispatching rules can be implemented such that a feasible schedule for both the AGVs and ASCs is guaranteed.

5.3.2 Deadlocks

Although the dispatching rules in the previous section seem to be generally applicable, the specific setting of a container terminal as considered in this thesis, 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 AGVs and ASCs are occupied and cannot be released of their containers. ASCs wait for empty AGVs, which are not available, since the loaded AGVs wait for a container that should precede at the QC, but which cannot be transported since all AGVs are occupied. Recall Example 1, as discussed in Subsection 3.5.1. In this subsection, a simple example is given for which the First Come First Served rule results in a deadlock situation. Although this example may seem somewhat artificial, it illustrates perfectly the complicated deadlocks that frequently occurred after implementing the dispatching rules in our detailed simulation model of the container terminal (see Section 3.3).

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 and 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 (AGVs and ASCs 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 5.3.1 in such a way that they always guarantee deadlock free schedules.

5.3.3 Deadlock free dispatching rules

In this subsection, we will discuss how the dispatching rules as mentioned in Subsection 5.3.1 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 ASCs, AGVs and QCs. For a schedule to be feasible, the following two conditions are sufficient:*

1. *The order of the containers on the ASCs and the AGVs coincide, i.e., for each ASC s, the order π_s in which the containers are handled, is a suborder of π, the assignment order of the containers to the AGVs.*
2. *The order of the containers on the QCs and the AGVs 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 schedule π result in 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 AGVs 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 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 assignment of the containers to the idle AGVs. The order of the containers on the ASCs is kept in accordance with this AGV assignment. So, Condition 1 of Theorem 5.1 is always satisfied.

1. Nearest Vehicle/Workstation First. This rule cannot be applied within the specific setting of container terminals we consider here. Recall the layout of Figure 4.1 in which all AGVs pass a common point after unloading. So, there is

in fact only a single delivery location. Applying this rule will assign all AGVs to containers that are located in the nearest stack lane. Obviously, this will lead to deadlock situations since this dispatching rule does not respect the order of the containers at the QCs 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 (and is available longest). 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 the order in which the containers are assigned to the AGVs corresponds to the order in which a QC handles its containers.

4. Fixed Assignment of AGVs to QCs. 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^{agv} (see equation (4.11)) 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 containers

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 container j. This is obvious whenever we have a perfect loop layout (the time required to handle a container starting from the common point, to the stack, to the QC and back to the common point, is the same for all containers). However, in case we only have a common point that all AGVs pass, 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 container j may be smaller than the due date of container i (if containers i and j are in different stack lanes). If this is the case, we force the due date of container j to be slightly larger than the due date of container i. In this way, we ensure Condition 2 of Theorem 5.1 holds. Note that each time a container is loaded by one of the QCs, 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, may 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 AGVs handle their containers in accordance to the order of the containers on the ASCs (Condition 1 of Theorem 5.1).

5.4 Dispatching rules versus Beam Search

In this section, we will discuss our computational study to investigate the performance of the various dispatching rules as discussed in Subsection 5.3.3. 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 by implementing them in the simulation model, as discussed in Section 5.2. Moreover, the same loading operations (over 150 operations) were generated as for the results discussed in Section 5.2. 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 5.3.1, were not considered since they are all subordinate to the minimization of the makespan. Table 5.5 gives the average deviation of 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 5.1 to 5.3).

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)
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)
Most Work Remaining	2.4	(3.3)	2.2	(3.2)	1.9	(3.2)
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 5.5: 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 5.5 we give results for both a horizon of 10 and 50 containers per QC. The more containers that are within the planning horizon, the less likely it is that an ASC becomes idle, since if there is a container available it can already start handling this container.

2. For the Fixed Assignment rule, the numbers in Table 5.5 are only based on cases in which the number of AGVs is a multiple of the number of QCs, since in the Fixed Assignment rule, each QC has a fixed number of AGVs assigned to it. So, we have that for each QC and equal number of AGVs is working.

The results in Table 5.5 give rise to the following conclusions. The performance of the modified dispatching rules varies a lot and the dynamic version of the Beam Search algorithm still performs best. The dispatching rules that give quite satisfactory results are the Most Work Remaining rule and the First Come First Served rule with a large planning horizon. These two rules take the information about future containers into account. Note that their good performance again confirms our conclusion from Section 5.2 that taking information about future containers into account, is more important than reacting on the current situation, which is a result of the

partial realization of the schedule. The Earliest Due Date rule, for instance, reacts on the 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 our computational testing we learned that not taking these conditions into account, for instance, by defining dispatching rules which are more "loose", will often lead to deadlock situations. Note that the conditions of Theorem 5.1 are sufficient to obtain a feasible schedule; they are not proven to be necessary.

Another conclusion we can draw from Table 5.5 is that the dispatching rules tend to perform slightly better, relative to the Beam Search, as the stochasticity increases.

5.5 Conclusions

In this chapter, 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 problem can be solved by using for instance the Beam Search algorithm discussed in Chapter 4. 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 the dynamic version of the Beam Search algorithm and compared it to the performance of the static version of the algorithm. In the static version, we solve the whole problem at once. In the dynamic version, a rolling horizon is used and rescheduling is done after certain fixed time intervals. 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 crucial to achieve satisfactory performance, i.e., a performance which is relatively close to the performance of the static version of the algorithm. This result holds both for the deterministic and for the stochastic scenarios. Hence, it seems that taking information of future containers into account, although note completely reliable, is advantageous.

The second part of the chapter considered various dispatching rules. It is said that such rules are very general and easy to implement. Therefore 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.

Since the performance of the static version of the Beam Search algorithm is very good, a good strategy may be to schedule all containers at the beginning of the loading operation and to execute this schedule. Rescheduling is done only in case of major breakdowns. Whenever rescheduling is necessary, we take the planning horizon as large as possible.

Chapter 6

An integer programming approach to integrated scheduling of handling equipment

In the previous two chapters, we have considered the scheduling of handling equipment for a specific layout of the container terminal in which all AGVs pass a common point after unloading by the QCs. In this chapter, we will present a model for a general terminal layout. The problem of scheduling ASCs, AGVs and QCs is now formulated as a Mixed Integer Program (MIP). Based on this formulation, we derive some classes of valid inequalities. The effectiveness of the inequalities are tested in some preliminary computational experiments.

The remainder of this chapter is organized as follows. In Section 6.1, we will discuss the more general context of the scheduling problem of ASCs, AGVs and QCs, followed in Section 6.2 by some modeling issues and the complexity of the problem. Next, in Section 6.3, we model the integrated scheduling problem as a Mixed Integer Program (MIP). In Sections 6.4 to 6.7, we will derive some valid inequalities for this formulation. Associated with these valid inequalities are the separation problems, which are discussed in Section 6.8. In Section 6.9 we will discuss some preprocessing rules and logical constraints. On the computational results, we will report in Section 6.10. Finally, we will make some concluding remarks in Section 6.11.

6.1 Problem definition

At the current generation of automated container terminals in Rotterdam, the AGVs drive in a loop layout, all passing a common point after they have been unloaded at the QC (see Figure 4.1 in Chapter 4). However, the disadvantage of such a layout is that the AGVs have to drive relatively far, whereas it would be more efficient to let them drive directly from the stack lane to the QC. Shorter driving distances will reduce the number of AGVs required. Moreover, since there are fewer vehicles, there is generally less interference between the AGVs and thus, the traffic control is easier (see also Subsection 2.4.3).

One possible change of the layout of the AGV area is to allow the AGVs to take shortcuts. For instance, after an AGV has been unloaded by a QC, it does not have to follow the loop, but it may cross the middle area directly after the QC. To do so, there should be enough space between the QCs. In this way, the driving distance to the stack lane where the AGV should pick up its next container can be reduced. A layout with shortcuts is illustrated in Figure 6.1. The shaded AGVs are loaded, the transparant AGVs are empty. Note that between the two most right QCs there is not enough space and thus, the AGVs cannot take shortcuts.

Evers *et al.* (1997) also present an alternative layout for automated container terminals. This layout is illustrated in Figure 6.2. In this layout, the loaded AGVs cross the middle area on their way from the stack to the QCs. A loaded AGV first drives onto a turning platform, which turns the AGV parallel to the quay. After the container is lifted off by the QC, the empty AGV drives to the stack using one of the outer loops (either right or left).

The layouts as presented in Figures 6.1 and 6.2 are more general than the one presented in Figure 4.1 of Chapter 4, which represents the current generation of terminals. In the next section, we will model the scheduling problem of equipment at the more general terminal.

6.2 Modeling and complexity

In this section, we will model the integrated scheduling problem of S ASCs, M AGVs and Q QCs, for a general layout of the AGV area, as a Mixed Integer Program (MIP). Apart from the difference in layout, we make the same assumptions for the scheduling problem as in Chapters 3 and 4. Before we give the formulation, we will first focus on some important modeling issues and discuss the complexity of the problem.

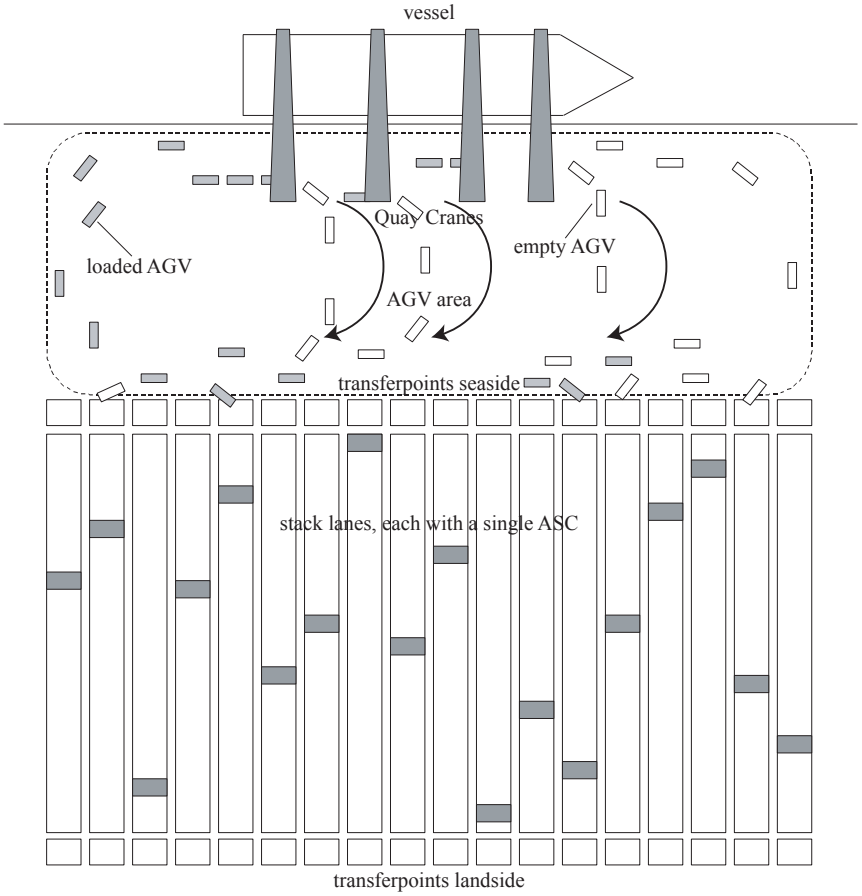

Figure 6.1: Terminal layout with shortcuts

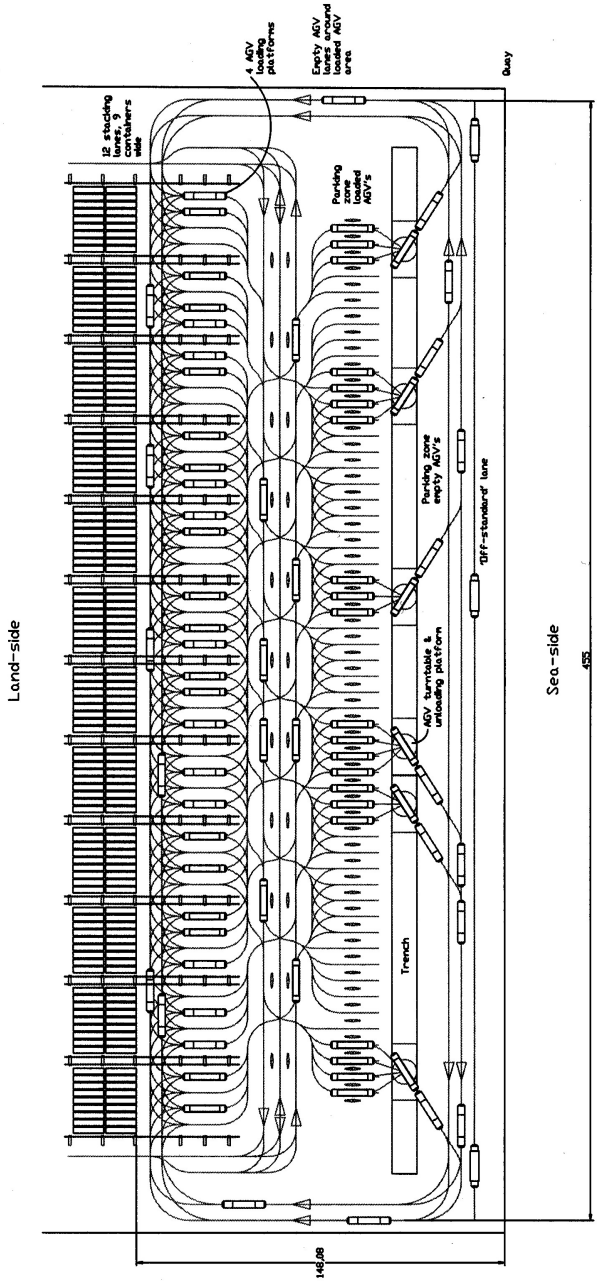

Figure 6.2: Alternative layout with direct transport from stack to quay (source: Evers *et al.* (1997))

6.2.1 Modeling

As in Chapter 4, we associate the loading of each container with three tasks: the ASC, AGV and QC task. Throughout this chapter, we will often refer to the ASC/AGV/QC task related to container i simply as (ASC/AGV/QC) task i. The more general layout of the AGV area, as illustrated in Figures 6.1 and 6.2 can be modeled by introducing for each AGV task a sequence dependent switch-over time. For instance, if an AGV transports containers i and j directly after each other, there is a sequence dependent switch-over time s_{ij}^{agv}. This time represents the driving time of the AGV from the QC at which container i is unloaded, to the stack lane at which container j is picked up. Note that in the model as presented in Chapter 4, we introduced a setup time which only depends on the next container j, since each AGV passes a common point after it is unloaded. That is, at the common point, all AGVs can be considered the same, i.e., the previous task of an AGV has no influence.

For the remaining, we use the same assumptions as presented in Chapter 3. Moreover, similar to the problem which was discussed in Chapter 4, we assume to have blocking constraints (Section 4.2.1), time-lags (Section 4.2.2) and the one-to-one correspondence between the load sequence of the QCs and the stack (Section 4.2.3). Moreover, we again assume that handling times are known and preemption is not allowed.

6.2.2 Complexity

The problem as described in this section clearly generalizes the scheduling problem as discussed in Chapter 4 and as a result, is NP-hard as well. For more details, we refer to Theorem 4.1.

6.3 Formulation

In this section, we will formulate the integrated scheduling problem of ASCs, AGVs and QCs as a Mixed Integer Program (MIP). First, we introduce the following parameters:

N	:	the number of containers to be loaded
M	:	the set of automated guided vehicles (AGVs)
S	:	the set of automated stacking cranes (ASCs)
$N(s)$	:	the set of containers to be handled by ASC $s \in S$
$\sigma(i)$	:	the set of successors of container i on the QC
$\pi(i)$	:	the set of predecessors of container i on the QC
$p_i^{qc/agv/asc}$	:	handling time of QC/AGV/ASC task i
s_{ij}^{agv}	:	switch-over time between AGV tasks i and j
K	:	sufficiently large number

Next, we define the variables. We introduce binary variables to represent the order of the containers on the equipment. It is possible to use 0/1 variables with different interpretations (see Queyranne and Schulz (1994) for more details). In our model we will use the so-called TSP variables, which are only set to 1 if two containers are handled *directly* after each other. An alternative would be to use 0/1 variables that only indicate if two containers are scheduled after another or not (so, not necessarily directly after another).

$$x_{ij} = \begin{cases} 1 & \text{if container } j \text{ is handled } \textit{directly} \text{ after container } i \text{ by same the ASC} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if container } j \text{ is handled } \textit{directly} \text{ after container } i \text{ by the same AGV} \\ 0 & \text{otherwise} \end{cases}$$

Next to the 0/1 variables, we also define continuous variables to represent the completion times of tasks.

$$\begin{aligned} b_i &= \text{completion time of ASC task } i \\ c_i &= \text{completion time of AGV task } i \\ C_{max} &= \text{completion time of the last container handled by any QC} \end{aligned}$$

Note that for the QCs, we define no decision variables, except C_{max}. Recall from Section 4.2.2 that the QCs can be modeled by time-lags on the AGV tasks. We can now define model the integrated scheduling problem as follows:

$$\text{Min} \quad C_{max} \tag{6.1}$$

s.t.

$$\sum_{\substack{j \in N \\ j \neq i}} x_{ij} = 1 \qquad \forall i \in N \tag{6.2}$$

$$\sum_{\substack{j \in N \\ j \neq i}} x_{ji} = 1 \qquad \forall i \in N \tag{6.3}$$

$$\sum_{\substack{j \in N \\ j \neq i \\ j \notin \pi(i)}} y_{ij} = 1 \qquad \forall i \in N \tag{6.4}$$

$$\sum_{\substack{j \in N \\ j \neq i \\ j \neq \sigma(i)}} y_{ji} = 1 \qquad \forall i \in N \tag{6.5}$$

$$K(1 - x_{ij}) + b_j - b_i \geq p_j^{asc} \qquad \forall i, j \in N \tag{6.6}$$

$$K(1 - y_{ij}) + b_j - c_i \geq s_{ij}^{agv} \qquad \forall i, j \in N \tag{6.7}$$

$$c_i \geq b_i + p_i^{agv} \qquad \forall i \in N \tag{6.8}$$

$$c_j \geq c_i + p_i^{qc} \qquad \forall i, j \in N : \sigma(i) \setminus \sigma(j) := \{j\} \tag{6.9}$$

$$C_{max} \geq c_i + p_i^{qc} \qquad \forall i \in N : \sigma(i) = \emptyset \tag{6.10}$$

$$x_{ij} \in \{0, 1\} \qquad \forall s \in S, \forall i, j \in N(s) \tag{6.11}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{6.12}$$

The objective (6.1) is to minimize the makespan, that is, to minimize the loading time of the ship. Clearly, the ship is loaded whenever the last container is loaded. To be more specific, the makespan is greater than or equal to the completion time of an AGV task related to a container that is handled last ($\sigma(i) = \emptyset$) on a QC, plus the corresponding handling time of this container on the QC. This is modeled by equation (6.10).

The degree constraints, that is, equations (6.2) to (6.5), state that each container handled on an ASC or AGV is succeeded and preceeded by exactly one other container. Therefore, we introduce for each AGV and each ASC two dummy containers. One dummy representing the first and the other representing the last container handled.

The difference between the completion times of two containers handled successively by the same ASC has to satisfy (6.6), which states that this difference should be at least equal to the handling time of the latter. Note that this restriction also prevents the existence of subtours among ASC tasks, that is, we cannot have a cyclic

order in which $x_{ij}, x_{jk}, \ldots, x_{li}$ are all set to 1.

The blocking of the ASC by the AGV, as discussed in Subsection 4.2.1 is modeled by restriction (6.7), which says the following. Suppose some AGV transports containers i and j directly after each other. Then the AGV will finish task i at time c_i. Next, some switch-over time is needed before the AGV can start transporting container j, i.e., the AGV has to drive to the proper ASC to pick up container j. Hence, the ASC cannot complete task j earlier than the time at which the AGV arrives. As a result, the completion time on the ASC of task j should be greater than or equal to the completion time on the AGV of task i, plus the switch-over time s_{ij}^{agv}. Similarly like (6.6), this restriction also rules out subtours on AGV tasks.

The completion time of an AGV task is always greater than or equal to the completion time of the corresponding task at the ASC (by definition the starting time of the AGV task), plus the handling time of the container on the AGV. This is modeled by (6.8).

The time-lags, as discussed in Section 4.2.2, which represent the handling of the containers by the QCs, are modeled by restriction (6.9). So, for two containers i and j that are handled by the same QC, such that j is the immediate successor of container i on the QC, we have that the completion time of AGV task j is greater than or equal to the completion time of AGV task i, plus p_i^{qc}: the handling time of container i on the QC. This is due to the blocking of the AGV by the QC as discussed in Subsection 4.2.1.

Finally, restrictions (6.11) and (6.12) define integrality of the binary decision variables. Note that the variables x_{ij} are only defined for containers i and j that are stored in the same stack lane and thus are to be handled by the same ASC. Moreover, for containers i, j such that j precedes i on the QC, we can set $y_{ij} = 0$, since setting $y_{ij} = 1$ would imply that container i is handled before container j on the QC (this follows directly from (6.7) and (6.8)).

In the next sections, we will derive several classes of valid inequalities for the formulation given by equations (6.1) to (6.12).

6.4 Cycle and clique inequalities

In this section, we will discuss the cycle and clique inequalities. These inequalities consist of binary variables x_{ij} and y_{ij} only. Furthermore, we will discuss lifted versions of these inequalities.

6.4.1 Basic cycle and clique inequalities

In Theorem 6.1, the cycle inequalities are defined which consist of both x_{ij} and y_{ij} variables:

Theorem 6.1 *Suppose we have K containers, $K := \{1, \ldots, k\}$. Moreover, define $k + 1 = 1$. Now let $S \subseteq K$ be an arbitrary subset of the containers and $\bar{S} := K \setminus S$ the complement. Then we have the following valid inequality:*

$$\sum_{i \in S} x_{i,i+1} + \sum_{i \in \bar{S}} y_{i,i+1} \leq |K| - 1 \tag{6.13}$$

Proof: Recall from our MIP formulation, equations (6.7) and (6.8). By substituting (6.8) into (6.7) we obtain:

$$K(1 - y_{ij}) + b_j - b_i \geq p_i^{agv} + s_{ij}^{agv} \tag{6.14}$$

So, by equation (6.14) we have that $y_{i,i+1} = 1$ implies $b_i < b_{i+1}$. Moreover, from equation (6.6) it follows directly that setting some $x_{i,i+1} = 1$ gives $b_i < b_{i+1}$. Hence, setting all variables in the cycle equal to 1, will give $b_1 < b_k$ and $b_k < b_1$, a contradiction. □

Next, we define the mixed clique inequalities, i.e., clique inequalities that involve both ASC and AGV tasks.

Theorem 6.2 *Suppose we have K containers, $K := \{1, \ldots, k\}$. Now let $C_1 \subseteq K$ be an arbitrary subset of the containers and let C_2 be the complement. Then the following inequality is valid:*

$$\sum_{i \in C_1} \sum_{j \in K} x_{ji} + \sum_{i \in C_2} \sum_{j \in K} y_{ji} \leq |K| - 1 \tag{6.15}$$

So, set C_1 is associated with ASC tasks (x_{ji}) and set C_2 is associated with AGV tasks (y_{ji})

Proof: Suppose the inequality does not hold, i.e., we can set at least $|K|$ variables equal to 1. From the degree constraints (6.3) and (6.5) and the definition of the inequality, it follows then that the sum of the incoming variables is at most 1 for each task. Since we have to set at least $|K|$ variables equal to 1, and we only have $|K|$ tasks, this means that each task has exactly one incoming variable equal to 1. But this means that we must have a (mixed) cycle, which is not possible by Theorem 6.1. □

6.4.2 Lifting cycle inequalities

Lifting, cf. Nemhauser and Wolsey (1988), is a technique to strengthen valid inequalities. In this section, we will apply this technique to the cycle inequalities as defined in the previous section.

D_k inequalities

Assume we have a cycle consisting of variables x_{ij} or y_{ij} only. By sequentially lifting additional variables, we can obtain various classes of inequalities. Well known classes are the so-called D_k^- and D_k^+ inequalities, which hold for the Asymmetric Traveling Salesman Problem (ATSP), cf. Grötschel and Padberg (1977).

Starting with a cycle inequality on $|K|$ tasks, sequential lifting gives us (among others) the D_k^- and D_k^+ inequalities which are defined by equations (6.16) and (6.17) respectively.

$$\sum_{i=1}^{k} y_{i,i+1} + 2\sum_{i=3}^{k} y_{1,i} + \sum_{i=4}^{k}\sum_{j=3}^{i-1} y_{i\,j} \leq |K| - 1 \tag{6.16}$$

$$\sum_{i=1}^{k} y_{i,i+1} + 2\sum_{i=2}^{k-1} y_{i,1} + \sum_{i=3}^{k-1}\sum_{j=2}^{i-1} y_{i\,j} \leq |K| - 1 \tag{6.17}$$

An example of a D_k^- inequality is given in Figure 6.3.

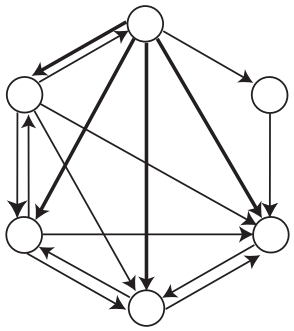

Figure 6.3: D_k^- inequality for $|K| = 6$

Clearly, the D_k inequalities are valid. We will only give a sketch of the proof, based on Figure 6.3. Suppose we select none of the variables (arcs) with coefficient 2. Then we should select $|K|$ arcs, such that the degree constraints (indegree and outdegree at most one) hold. Hence, we must have a cycle, which cannot be the case. If we

do select a variable with coefficient 2, we should select another $|K-2|$ arcs. Taking again the degree constraints into account, we should have a path connecting all $|K|$ nodes. However, the only possible predecessor in the path of node 2 is node 1. But since we have chosen an arc with coefficient 2, which is an outgoing arc of node 1, this cannot be the case anymore.

Note that in general, the number of different lifted cycle inequalities can grow fast if K (the length of the cycle) increases.

Other classes of inequalities that are valid for the ATSP, and are also valid for the scheduling problem we consider, were introduced by Balas and Fischetti (1999). These are the shell, fork and curtain inequalities.

6.5 Precedence based inequalities

In this section, we will discuss some classes of valid inequalities that can be derived by using the precedence constraints among the containers to be handled on the same QC. As in the previous section, we will focus only on inequalities that involve binary variables.

Let N_q denote the set of AGV tasks related to the containers to be handled by QC q. Recall from Section 4.2.2 that the order in which the containers are handled by a single QC is fixed and that this order can be translated to time-lags on the corresponding AGV tasks. Moreover, these time-lags are chain-like. For the remainder of this section, we will treat the time-lags as if they were ordinary chain-like precedence constraints, as illustrated in Figure 6.4. Note that each AGV task is in exactly one chain. Now define $i \prec j$ if container i precedes container j on the

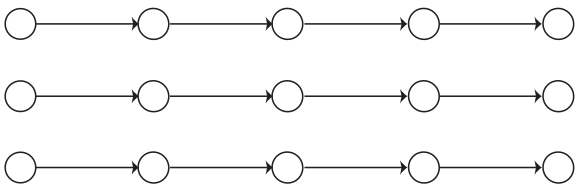

Figure 6.4: Chain-like precedence constraints

same QC, and so we have that $c_i < c_j$ by equation (6.9). Note that this does not necessarily imply $b_i < b_j$, even if containers i and j are related to the same ASC. Moreover, define N_q^+ as the set of tasks $i_q \in N_q$ and all its successors in the chain,

that is:

$$N_q^+ := \{i \in N_q | i \succeq i_q\} \tag{6.18}$$

Similarly, we define N_q^- as the set consisting of task $i_q \in N_q$ and all its predecessors in the chain:

$$N_q^- := \{i \in N_q : i \preceq i_q\} \tag{6.19}$$

For notational convenience, we define the following parameter, which represents the precedence constraints among the containers on the QC:

$$z_{ij} = \begin{cases} 1 & \text{if container } i \text{ precedes container } j \text{ on the same QC} \\ 0 & \text{otherwise} \end{cases}$$

Note that, because of transitivity, we have that if $z_{ij} = 1$ and $z_{jk} = 1$, this implies $z_{ik} = 1$. Next, we will derive some properties which are induced by the precedence constraints (z_{ij}'s).

Property 6.5.1 *If $x_{ij} = 1 \wedge z_{jk} = 1$ or $y_{ij} = 1 \wedge z_{jk} = 1$, then $b_i < b_j < c_k$. That is, whenever task i and j are scheduled consecutively on the same ASC or AGV, and container j must precede container k on the QC, then the completion time of ASC task i is smaller than the completion time of AGV task k.*

Proof: From equation (6.14) (see Theorem 6.1), it follows that $b_i < b_j$ if $x_{ij} = 1$ or $y_{ij} = 1$. Combining this result with equations (6.8) and (6.9), gives us $b_j < c_j < c_k$. □

Property 6.5.2 *If $z_{ij} = 1 \wedge y_{jk} = 1$, then $b_i < c_i < c_j < b_k$. So, if container i precedes container j on the QC, and AGV tasks j and k are scheduled consecutively on the same AGV, then the completion time of ASC task i is smaller than the completion time of ASC task k.*

Proof: From equation (6.8) it follows immediately that $b_i < c_i$. Moreover, since $z_{ij} = 1$ we have $c_i < c_j$. Finally, from (6.7) it follows immediately that $c_j < b_k$. □

A similar property for the case $z_{ij} = 1 \wedge x_{jk} = 1$ cannot be given. Although $z_{ij} = 1$ implies $b_i < c_i < c_j$ and $x_{jk} = 1$ implies $b_j < b_k$, this does not necessarily result in $b_i < b_k$.

In the next sections, we will discuss several classes of valid inequalities that can be derived, using the precedence constraints, represented by the parameters z_{ij}.

To facilitate the exposition, the parameters z_{ij} appear on the left hand side of the equations. Clearly, they can be moved to the right hand side, decreasing the right hand side with their value.

6.5.1 Precedence based cycle inequalities

The first class of precedence based inequalities are generalizations of the (mixed) cycle inequalities, as defined in Section 6.4. These generalized cycle inequalities, will satisfy the conditions of Properties 6.5.1 and 6.5.2. Note that we only have to consider generalized cycle inequalities in which we have no consecutive z_{ij}'s, since in that case we can replace them with a single z_{ij} and decrease the right hand side by 1.

Theorem 6.3 *Consider K containers, $K := \{1 \dots k\}$ and define $k+1 = 1$. Moreover, let S_1, S_2 and S_3 be a partition of K with the property that if $i \in S_3$, then $i+1 \notin S_1$. Then the following inequality is valid:*

$$\sum_{i \in S_1} x_{i,i+1} + \sum_{i \in S_2} y_{i,i+1} + \sum_{i \in S_3} z_{i,i+1} \leq |K| - 1 \tag{6.20}$$

That is, any generalized cycle inequality that consists of arbitrary variables x_{ij}, y_{ij} and parameters z_{ij}, all equal to 1, is valid provided that no z_{ij} is immediately succeeded by some x_{jk}.

Proof: First note that if not all z_{ij} are equal to 1, the inequality is trivial. So given all z_{ij} equal to 1, the inequality implies that some x_{ij} or y_{ij} should be set to 0.

From the proof of Theorem 6.1, it follows that $x_{ij} = 1 \Rightarrow b_i < b_j$ and $y_{ij} = 1 \Rightarrow b_i < b_j$. Besides, $z_{i,i+1} = 1 \wedge y_{i+1,i+2} = 1$ for some $i \in S_2$ implies $b_i < b_{i+2}$ (Property 6.5.2). Note that any generalized cycle inequality that consists of z_{ij}'s only is not valid by definition, and that cycle inequalities with no z_{ij}'s are valid by Theorem 6.1. Hence, w.l.o.g. we may assume that y_{12} is the first variable in the cycle inequality and z_{k1} is the last parameter in the inequality, since by definition of the inequality, we cannot have that z_{k1} precedes x_{12} in the cycle. By successively applying Properties 6.5.1 and 6.5.2, we obtain $b_1 < b_2 < b_k$. Moreover, $b_k < b_2$ (Property 6.5.2), which yields a contradiction. □

It follows from Theorem 6.3 that we can generalize the mixed cycle inequalities by using the parameters z_{ij}, provided that z_{ij} is not succeeded by some x_{jk}. This implies the following:

Corollary 6.4 *Given a set of K containers, $K := \{1 \dots k\}$ and $k+1 = 1$ and a partition S_1, S_2 of K, the following inequality is valid:*

$$\sum_{i \in S_1} y_{i,i+1} + \sum_{i \in S_2} z_{i,i+1} \leq |K| - 1 \tag{6.21}$$

I.e., any cycle consisting of y_{ij} 's and z_{ij} 's, all equal to 1, is not allowed.

Proof: Follows directly from Theorem 6.3 by taking S_1 in equation (6.20) empty. □

6.5.2 Extended cycle inequalities

In this section, we will extend the general cycle inequalities (6.20) by introducing additional variables that are related to successors and/or predecessors of containers in the cycle K. Suppose we have a variable y_{ij} in our general cycle inequality (6.20). Since the degree constraints (equations (6.4) and (6.5)) hold, we can introduce the variables y_{kj} such that $k \succ i$ into the inequality, without increasing the right hand side. In more general form, we obtain the following:

Theorem 6.5 *Consider a set of containers K, $K := \{1 \dots k\}$ and $k+1 = 1$. Moreover, let S_1, S_2, S_3, S_4 and S_5 be a partition of K with the properties:*

1. *If $i \in S_2 \cup S_4 \cup S_5$ then $i+1 \notin S_1 \cup S_2$.*
2. *If $i \in S_5$ then $i+1 \notin S_3$.*
3. *If $i \in S_4$ then $i+1 \notin S_5$.*
4. *If $i \in S_1$ then $i+1 \notin S_3 \cup S_4 \cup S_5$.*

Then the following inequality is valid [1].

$$\begin{aligned} \sum_{i_q \in S_1} x_{i_q,i_q+1} + \sum_{i_q \in S_2} \sum_{j \in N^-_{q+1}} x_{i_q,j} + \sum_{i_q \in S_3} \sum_{j \in N^+_q} y_{j,i_q+1} \\ + \sum_{i_q \in S_4} \sum_{j \in N^-_{q+1}} y_{i_q,j} + \sum_{i_q \in S_5} z_{i_q,i_q+1} \quad \leq \quad |K| - 1 \end{aligned} \tag{6.22}$$

Proof: Assume the inequality does not hold. Then we can obtain a left hand side of at least K. Consider two successive containers i_q and $i_q + 1$. Clearly, if $i_q \in S_1$,

[1] Note that the first property is necessary to obtain a valid inequality, the other properties only imply stronger inequalities

we have $x_{i_q,i_q+1} = 1$ and if $i_q \in S_5$ we have $z_{i_q,i_q+1} = 1$. Moreover, for each $i_q \in S_2$ we should have:

$$\sum_{j \in N^-_{q+1}} x_{i_q,j} \leq 1 \tag{6.23}$$

because of the degree constraint on i_q. A similar condition holds for all containers $i_q \in S_4$. Finally, for all jobs $i_q \in S_3$ we have:

$$\sum_{j \in N^+_q} y_{j,i_q+1} \leq 1 \tag{6.24}$$

because of the degree constraint on $i_q + 1$. Since we have $|K|$ containers in our cycle, setting $|K|$ variables equal to 1, would result in a cycle as in Theorem 6.3, which is not valid. □

An example of an extended cycle inequality (6.22) is given in Figure 6.5.

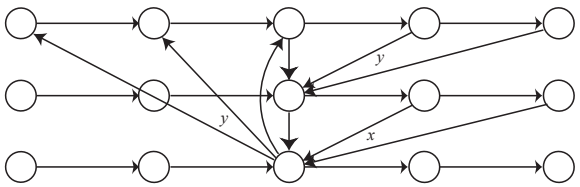

Figure 6.5: Extended cycle inequality

Next, we will show that the extended cycle inequalities, dominate the inequalities given by (6.20).

Theorem 6.6 *The extended cycle inequalities dominate the inequalities given by (6.20).*

Proof: Consider the extended cycle inequality (6.22). Now take $N^-_q = \emptyset$ and $N^+_q = \emptyset$. As a result, we obtain the following inequality:

$$\sum_{i_q \in S_1 \cup S_2} x_{i_q,i_q+1} + \sum_{i_q \in S_3 \cup S_4} y_{i_q,i_q+1} + \sum_{i_q \in S_5} z_{i_q,i_q+1} \leq |K| - 1 \tag{6.25}$$

which is equivalent to (6.20). □

6.5.3 Cut set inequalities

In this section we will give another class of inequalities which is valid for the binary variables y_{ij} only.

Let Q be a subset of the QCs and w.l.o.g., assume $Q := \{1, \ldots, |Q|\}$. Recall the definition of N_q, that is, N_q is the set of AGV tasks corresponding to the containers to be handled by QC q. Now define a cut set $C := \{i_1, \ldots, i_{|Q|}\}$ as a set of AGV tasks such that for each set $N_q : q \in Q$, we select exactly one task i_q in the cut set. Moreover, let N_q^+ and N_q^- be as defined before. A cut set is illustrated in Figure 6.6. We can now construct the following valid inequalities:

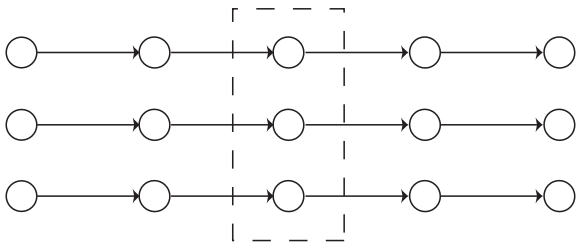

Figure 6.6: A cut set

Theorem 6.7 *Given a subset Q of QCs, and a cut set $C := \{i_1, \ldots, i_{|Q|}\}$, the following inequalities are valid:*

$$\sum_{\substack{q \in Q \\ q \neq 1}} \sum_{j \in N_q^+} y_{j\, i_1} + \ldots + \sum_{\substack{q \in Q \\ q \neq |Q|}} \sum_{j \in N_q^+} y_{j\, i_{|Q|}} \leq |C| - 1 \tag{6.26}$$

$$\sum_{\substack{q \in Q \\ q \neq 1}} \sum_{j \in N_q^-} y_{i_1 j} + \ldots + \sum_{\substack{q \in Q \\ q \neq |Q|}} \sum_{j \in N_q^-} y_{i_{|Q|} j} \leq |C| - 1 \tag{6.27}$$

Proof: We will prove only the validity of equation (6.26), the proof of the validity of (6.27) is similar. Suppose we can set $|C|$ variables equal to 1. Since the degree constraint holds, i.e., equation (6.5), this means that for every $i_q \in C$ we must have:

$$\sum_{\substack{p \in Q \\ p \neq q}} \sum_{j \in N_p^+} y_{j\, i_q} = 1 \tag{6.28}$$

Consider the variable equal to 1, associated with task $i_q \in C$. By definition, this variable is originating from some $i_p^+ \in N_p^+, p \in Q$. Hence, we have $y_{i_p^+, i_q} = 1$ and moreover, $z_{i_p, i_p^+} = 1$ with $i_p \in C$. So, we have a path from $i_p \in C$ to $i_q \in C$. Next, for i_p we can repeat the same arguments, and we obtain $z_{i_r, i_r^+} = 1 \wedge y_{i_r^+, i_p} = 1$, for

some $i_r \in C, i_r \neq i_q$. Hence, repeating another $|C|-2$ times, we must end up in some task $i_q \in C$ for the second time, since otherwise we would require at least $|C|+1$ tasks in the cut set, which is by definition not the case. But since we end up in a node $i_q \in C$ for the second time, we have a cycle as defined in Corollary 6.4, which cannot be the case. □

Theorem 6.8 *Equation (6.26) is satisfied with equality if and only if there is a $i_q \in C$ such that there is a path from i_q to all other nodes in the cut set, using only variables that are included in equation (6.26).*

Proof: Whenever we have a path from a node $i_q \in C$ to all other nodes in the cut set, we need to set at least $|C|-1$ variables to one. Since Theorem 6.7 holds, this means that we have exactly $|C|-1$ variables set to 1.

To see that the opposite statement holds, suppose that there is no such node i_q. Since (6.26) is satisfied with equality, and since the degree constraints hold (each node has at most one incoming variable), we should have a (sub)cycle, which is not possible by Corollary 6.4. □

The cut set inequalities are illustrated in Figure 6.7. Observe that inequality

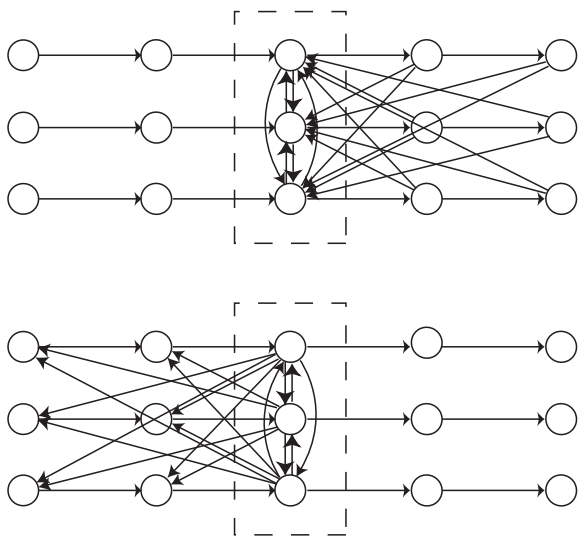

Figure 6.7: Cut set inequalities on successors and predecessors, respectively

(6.26) has the form of a clique inequality on the tasks $i_1, \ldots, i_{|Q|}$, extended with

all incoming variables of the successors of the tasks in the chain. We will use this observation in the separation of the inequalities as will be discussed in Section 6.8. Note that it is possible to enumerate all possible cut set inequalities. The number of different cut sets is $\mathcal{O}(n^{|Q|})$, where n is the total number of tasks and $|Q|$ is the total number of QCs. So, for a fixed number of QCs, there is a polynomial number of cut set inequalities. In practice, the number of QCs is quite low, i.e., 4 to 6.

6.5.4 Dominance of cut set inequalities

In this section, we will show that the cut set inequalities dominate the successor and predecessor inequalities, which are known from the literature on the Asymmetric Traveling Salesman Problem (ATSP), cf. Balas *et al.* (1995). Also the clique inequalities given by (6.15) in which $C_1 = \emptyset$ are dominated by the cut set inequalities, since these clique inequalities are already dominated by the successor inequalities.

Successor and predecessor inequalities

For the ATSP, Balas *et al.* (1995) define the successor and predecessor inequalities, which hold for general structures of precedence constraints. For the chain-like precedence constraints we consider, the cut set inequalities given by (6.26) and (6.27) form subsets of successor and predecessor inequalities. However, we will show that it is sufficient to consider only the cut set inequalities, since for any other violated successor or predecessor inequality, we can find a corresponding (violated) cut set inequality.

For notational convenience, define $x(S_1, S_2)$ as follows:

$$x(S_1, S_2) := \sum_{\substack{i \in S_1 \\ j \in S_2}} y_{ij} \tag{6.29}$$

Moreover, let $\sigma(S)$ and $\pi(S)$ denote respectively the set of successors and predecessors of the tasks in set S. Then we can rewrite the cut set inequalities (6.26) and (6.27) as follows:

$$x(C, C) + x(\sigma(C), C) \leq |C| - 1 \tag{6.30}$$

$$x(C, C) + x(C, \pi(C)) \leq |C| - 1 \tag{6.31}$$

The successor and predecessor inequalities are defined as:

$$x(S, S) + x(\bar{S} \cap \sigma(S), S) + x(\bar{S} \setminus \sigma(S), S \cap \sigma(S)) \leq |S| - 1 \tag{6.32}$$

$$x(S, S) + x(S, \bar{S} \cap \pi(S)) + x(S \cap \pi(S), \bar{S} \setminus \pi(S)) \leq |S| - 1 \tag{6.33}$$

In the next theorem, we will show that, in case we have chain-like precedence constraints, the cut set inequalities dominate the successor and predecessor inequalities.

Theorem 6.9 *Given chain-like precedence constraints, such that each task is exactly in one chain. Then the cut set inequalities defined by (6.30) and (6.31) dominate respectively the successor inequalities (6.32) and the predecessor inequalities (6.33).*

Proof: We will prove only the case of the successor inequality. The proof for the predecessor inequality is similar.

Note that we consider only chain-like precedence constraints. In case S is a cut set, i.e., S contains at most one task of each chain, we have that:

$$\bar{S} \cap \sigma(S) = \sigma(S) \tag{6.34}$$

and

$$S \cap \sigma(S) = \emptyset \tag{6.35}$$

Hence, in this case the successor inequality is similar to the cut set inequality (6.30).

The remainder of the proof considers the case when S is not a cut set, that is, there is a chain for which S contains more than one task. We will show that the successor inequality can be decomposed into a cut set inequality plus a number of degree constraints. Hence, whenever no cut set inequality is violated, it cannot be the case that a successor inequality is violated.

Now consider a successor inequality on set S and let C define a cut set in such a way that $C \subset S$ and for all $i \in C$ there is no task $j \in S$ such that $j \prec i$. That is, C is the subset of S such that the tasks in C have no predecessors within the set S. Distinguishing between C and $S \setminus C$, we obtain the following equivalent for the successor inequality (6.32):

$$\begin{aligned} & x(C, C) + x(S \setminus C,\ C) + x(S,\ S \setminus C) \\ & \quad + x(\bar{S} \cap \sigma(S),\ C) + x(\bar{S} \cap \sigma(S),\ S \setminus C) \\ & \qquad + x(\bar{S} \setminus \sigma(S),\ C \cap \sigma(S)) + x(\bar{S} \setminus \sigma(S),\ (S \setminus C) \cap \sigma(S)) \quad \leq \quad |S| - 1 \end{aligned} \tag{6.36}$$

Note that $x(\bar{S} \setminus \sigma(S),\ C \cap \sigma(S)) = 0$, since $C \cap \sigma(S) = \emptyset$ by the way we choose the cut set C. Moreover, we have that

$$\sigma(C) = \sigma(S) \tag{6.37}$$

as well as

$$\bar{S} \cap \sigma(S) = \sigma(C) \setminus (S \setminus C) \tag{6.38}$$

and

$$(S \setminus C) \cap \sigma(S) = S \setminus C \tag{6.39}$$

This leads to the following form of (6.36):

$$\begin{aligned} & x(C,C) + x(S \setminus C,\ C) + x(S,\ S \setminus C) \\ & + x(\sigma(C) \setminus (S \setminus C),\ C) + x(\sigma(C) \setminus (S \setminus C),\ S \setminus C) \\ & \qquad + x(\bar{S} \setminus \sigma(C),\ S \setminus C) \quad \leq \quad |S| - 1 \end{aligned} \tag{6.40}$$

The following terms of (6.40) imply no more than the degree constraints on the tasks in $S \setminus C$, i.e., we have that:

$$x(S,\ S \setminus C) + x(\sigma(C) \setminus (S \setminus C),\ S \setminus C) + x(\bar{S} \setminus \sigma(C),\ S \setminus C) \leq |S \setminus C| \tag{6.41}$$

As a result, a successor inequality (6.32) can only be violated whenever for the remaining part of (6.40) we have:

$$x(C,C) + x(S \setminus C,\ C) + x(\sigma(C) \setminus (S \setminus C),\ C) > |C| - 1 \tag{6.42}$$

Since C is chosen such that each task in $S \setminus C$ is a successor of some task in C, we have that:

$$(S \setminus C) \cup (\sigma(C) \setminus (S \setminus C)) = \sigma(C) \tag{6.43}$$

Hence, (6.42) reduces to:

$$x(C,C) + x(\sigma(C),\ C) > |C| - 1 \tag{6.44}$$

which implies the violation of a cut set inequality (6.26) on C. □

Balas *et al.* (1995) also define another class of valid inequalities, the so-called predecessor-successor inequalities. These inequalities are given for sets S_1 and S_2 such that $i \prec j$ for each pair of $i \in S_1, j \in S_2$. The chain-like precedence constraints only allow for such inequalities with $|S_1| = |S_2| = 1$. However, the inequalities are not valid for our problem, since they use implicitly the fact that a single salesman (a single AGV) visits all nodes. In our case however, we have multiple AGVs, and thus, multiple tours.

Clique inequalities

Recall from Section 6.4 that the clique inequality on a set $S \subseteq N$ of AGV tasks is given by:

$$\sum_{i \in S} \sum_{j \in S} y_{ji} \leq |S| - 1 \tag{6.45}$$

Following the notation as introduced in this subsection, this inequality can be rewritten as:

$$x(S,S) \leq |S| - 1 \tag{6.46}$$

Next, we state the following theorem about the dominance of the cut set inequalities over the clique inequalities:

Theorem 6.10 *The clique inequalities given by (6.45) are dominated by the cut set inequalities (6.26).*

Proof: It is obvious that the successor inequality as given by equation (6.32) dominates inequality (6.46). Since the successor inequality is dominated by the cut set inequality (6.30), it follows that the cut set inequality also dominates the clique inequality. □

Note that the result above only holds for clique inequalities consisting of variables related to AGV tasks. The clique inequalities for sets that contain ASC tasks, or a combination of AGV and ASC tasks, are not dominated by the cut set inequalities.

6.5.5 Mixed cut set inequalities

The cut set inequalities as defined so far, only consider AGV tasks. In this subsection, we will extend these inequalities to include also variables related with ASC tasks.

Now consider again a cut set inequality, which is slightly modified by deleting the variables among the tasks in the cut set. Let i_q^+ denote a task associated with some task that succeeds i_q at the QC, i.e., $z_{i_q i_q^+} = 1$. Now select a cut set C and a cut set C^+ of successors of C, that is, $C := \{i_1, \ldots, i_{|Q|}\}$ and $C^+ := \{i_1^+, \ldots, i_{|Q|}^+\}$. Moreover, redefine N_q^+ as:

$$N_q^+ := \{i \in N_q | i \succeq i_q^+\} \tag{6.47}$$

Then we can state the following:

Theorem 6.11 *Given $|Q|$ chains N_q, a cut set $C := \{i_1, \ldots, i_{|Q|}\}$ and a cut set $C^+ := \{i_1^+, \ldots, i_{|Q|}^+\}$, as defined above, the following inequality is valid:*

$$\begin{aligned} \sum_{\substack{q \in Q \\ q \neq 1}} x_{i_1, i_q^+} + \ldots + \sum_{\substack{q \in Q \\ q \neq |Q|}} x_{i_{|Q|}, i_q^+} \\ + \sum_{\substack{q \in Q \\ q \neq 1}} \sum_{j \in N_q^+} y_{j,i_1} + \ldots + \sum_{\substack{q \in Q \\ q \neq |Q|}} \sum_{j \in N_q^+} y_{j,i_{|Q|}} \quad \leq \quad 2\,|C| - 2 \end{aligned} \tag{6.48}$$

Proof: The proof is in line with the proof of Theorem 6.7. Suppose the inequality does not hold, i.e., we can set $2|C|-1$ variables equal to 1. Note that because of the degree constraints (6.2), we have that:

$$\sum_{\substack{q\in Q\\ q\neq 1}} x_{i_1,i_q^+} + \ldots + \sum_{\substack{q\in Q\\ q\neq |Q|}} x_{i_{|Q|},i_q^+} \leq |C| \tag{6.49}$$

Moreover, by Theorem 6.7 we have:

$$\sum_{\substack{q\in Q\\ q\neq 1}} \sum_{j\in N_q^+} y_{j,i_1} + \ldots + \sum_{\substack{q\in Q\\ q\neq |Q|}} \sum_{j\in N_q^+} y_{j,i_{|Q|}} \leq |C|-1 \tag{6.50}$$

Now, setting $2|C|-1$ variables equal to 1, implies that:

$$\sum_{\substack{q\in Q\\ q\neq 1}} x_{i_1,i_q^+} + \ldots + \sum_{\substack{q\in Q\\ q\neq |Q|}} x_{i_{|Q|},i_q^+} = |C| \tag{6.51}$$

and

$$\sum_{\substack{q\in Q\\ q\neq 1}} \sum_{j\in N_q^+} y_{j,i_1} + \ldots + \sum_{\substack{q\in Q\\ q\neq |Q|}} \sum_{j\in N_q^+} y_{j,i_{|Q|}} = |C|-1 \tag{6.52}$$

First, we consider the consequences of (6.51). Since the degree constraints hold, this implies for each individual task i_q^+ in C^+ that:

$$\sum_{i_p\in C, p\neq q} x_{i_p,i_q^+} = 1 \qquad \forall i_q^+ \in C^+ \tag{6.53}$$

So, for each task $i_q^+ \in C^+$, there is an incoming variable x_{i_p,i_q^+} set to 1.

The consequences of (6.52) follow directly from Theorem 6.8, that is, there is a node $i_q \in C$ for which we have a path to all other nodes in C. Obviously, by definition of the sets N_q^+, we have that all these paths pass node i_q^+. So, for i_q^+ we have a path to all nodes $i_p \in C, i_p \neq i_q$. Combining this with (6.53) gives us a cycle like (6.20), which by Theorem 6.3 cannot be the case. □

In Figure 6.8 we illustrate the mixed cut set inequality. The dotted arcs represent the x_{ij}'s. The solid arcs represent the y_{ij}'s.

6.6 Inequalities on completion times of tasks

In this section, we will discuss some inequalities for the continuous variables in our MIP formulation, i.e., the variables b_i, which represent the completion times of the ASC tasks.

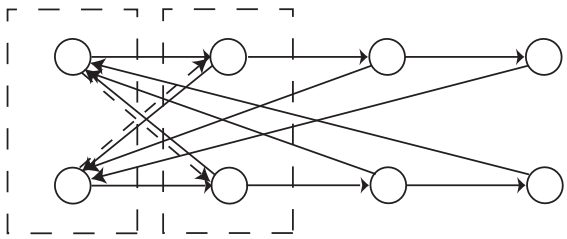

Figure 6.8: Extended cut set inequality

The scheduling of a single ASC can be viewed as a single machine scheduling problem. The inequalities we give in this section are only valid for tasks that are to be scheduled on the same ASC.

The following inequalities were introduced by Queyranne (1993) for the single machine scheduling problem. Consider a set of tasks S that is to be scheduled on the same ASC. Then the following inequality holds (for notational convenience denote p_i^{asc} by p_i):

$$\sum_{i \in S} p_i b_i \geq \sum_{i \in S} p_i^2 + \sum_{i \in S} \sum_{\substack{j \in S \\ j > i}} p_i p_j \tag{6.54}$$

For the simple problem in which we only have (positive) handling times and no release or due dates, these inequalities give a complete description of the convex hull for the single machine scheduling problem. Note that in this case we have a schedule with no idle time between the tasks. However, in the problem we consider, there might be idle time between two containers on the ASC, due to the limited availability of AGVs. So, it is obvious that equation (6.54) gives us a lower bound on the completion times of the ASC tasks. In a similar way, we can also derive an upper bound on the completion times, that is:

$$\sum_{i \in S} p_i b_i \leq C_{max} \cdot \sum_{i \in S} p_i - \sum_{i \in S} \sum_{\substack{j \in S \\ j > i}} p_i p_j \tag{6.55}$$

Note that on the right hand side C_{max} appears. Clearly, this can be replaced by any upper bound on C_{max}. However, every feasible solution gives such an upper bound. So, whenever a better solution is found during the solving of the problem, C_{max} is decreased and the inequalities (6.55) are strengthened.

Next, we will show how inequality (6.55) can be strengthened by using the fact that each ASC task is followed by the corresponding AGV task. Moreover, the AGV task is followed by the handling of the container on the QC, which has a number of

successors on the QC. Hence, for each ASC task, we can define a tail t_i^{asc} as follows:

$$t_i^{asc} = p_i^{agv} + \sum_{j \succeq i} p_j^{qc} \tag{6.56}$$

Given a value for C_{max}, an ASC task i has to be completed on time, taking into account its tail. So we should have that:

$$b_i \leq C_{max} - t_i^{asc} \tag{6.57}$$

So, the tail t_i^{asc} can then be interpreted as a due-date. This allows for the following strengthening of inequality (6.55), cf. Queyranne and Schulz (1995):

$$\sum_{i \in S} p_i b_i \leq (C_{max} - \min_{i \in S} t_i^{asc}) \cdot \sum_{i \in S} p_i - \sum_{i \in S} \sum_{\substack{j \in S \\ j > i}} p_i p_j \tag{6.58}$$

In order to extend inequality (6.54) in a similar way, a head h_i^{asc}, that is, the earliest time an ASC task can start, should be determined. Unfortunately, it is not possible to derive a similar expression like (6.56) for the h_i^{asc}'s in advance, i.e., initially $h_i^{asc} = 0$. Only if variables are set to one, it is possible to determine heads $h_i^{asc} > 0$, since the variables that are set to one define a partial order, that is, some of the predecessors of task i on the ASC may be set by this partial order.

6.6.1 Strengthening the subtour elimination constraints

In this section we show how we can strengthen the subtour elimination constraints of the original formulation, i.e. equations (6.6) and (6.7). Recall these equations from Section 6.3:

$$K(1 - x_{ij}) + b_j - b_i \geq p_j^{asc} \tag{6.59}$$

$$K(1 - y_{ij}) + b_j - c_i \geq s_{ij}^{agv} \tag{6.60}$$

The value of the parameter K should be sufficiently large in order to ensure validity of the inequalities in case x_{ij} or y_{ij} equals 0. On the other hand, we would like to choose the value of K as small as possible, since a large value of K results in a weak LP-relaxation. Whenever the values x_{ij} and y_{ij} are fractional in equations (6.59) and (6.60), and K is very large, the values of the continuous variables representing the completion times of the tasks do not have to differ much in order to satisfy the equations. As a result, the lower bound on the makespan we obtain is low.

Equations (6.59) and (6.60) can be strengthened by using the upper and lower bounds of the continuous variables that are included in the inequality. Hence, for specific tasks i and j we can introduce a parameter K_{ij}. Consider for instance

equation (6.59). We can then set the value of K_{ij} as the difference between the upper bound of variable b_i, minus the lower bound of variable b_j, plus the value of the right hand side. So, even if $x_{ij} = 0$, the inequality is satisfied. We will next determine the values of the K_{ij}'s more formally.

For notation purposes, relate K^1_{ij} to equations (6.59) and K^2_{ij} to equations (6.60). The values of the parameters are then defined as follows:

$$K^1_{ij} = C_{max} - p^{agv}_i - \sum_{k \succeq i} p^{qc}_k \tag{6.61}$$

$$K^2_{ij} = C_{max} - \sum_{k \succeq i} p^{qc}_k - p^{asc}_j + s^{agv}_{ij} \tag{6.62}$$

Note that for C_{max} we need initially an upper bound. A trivial bound is to sum up all handling times of all containers. However, if we find a better feasible solution during the optimization, C_{max} is decreased and also the K_{ij}'s can be updated accordingly. So we can replace equations (6.59) and (6.60) with:

$$K^1_{ij}(1 - x_{ij}) + b_j - b_i \geq p^{asc}_j \tag{6.63}$$

$$K^2_{ij}(1 - y_{ij}) + b_j - c_i \geq s^{agv}_{ij} \tag{6.64}$$

Another way to improve equation (6.60), i.e. the subtour elimination constraint on the AGV tasks, is to make use of the precedence constraints. This can be done by taking next to y_{ij}, also all the variables associated with the successors of i in the chain into the inequality:

$$K^2_{ij}(1 - \sum_{k \succeq i} y_{kj}) + b_j - c_i \geq s^{agv}_{ij} \tag{6.65}$$

6.7 Path inequalities using precedence constraints

In this section we will derive some valid inequalities that are based on variables that represent paths between AGV tasks related to two containers which are handled by the same QC. These constraints combine both the binary and continuous variables. Recall from Section 6.3 the equations (6.7) and (6.8) of the MIP formulation. Combining these restrictions gives us:

$$K(1 - y_{ij}) + c_j - c_i \geq s^{agv}_{ij} + p^{agv}_j \tag{6.66}$$

Note that in case containers i and j are handled by the same QC, and $i \prec j$, we have that $c_j > c_i$ by inequality (6.9). Hence, we can rewrite (6.66) as follows:

$$c_j - c_i \geq y_{ij}(s^{agv}_{ij} + p^{agv}_j) \tag{6.67}$$

Equation (6.67) states that if an AGV transports container j directly after container i, then the difference in completion time of the tasks is at least the switch-over time plus the handling time. Note that (6.67) only holds whenever $i \prec j$. For notational convenience, we define the following parameter:

$$P_{ij}^{qc} = \sum_{\substack{k \succeq i \\ k \prec j}} p_k^{qc} \tag{6.68}$$

Parameter P_{ij}^{qc} represents the minimal difference between the completion time of task i and task j where container i precedes container j on the QC, i.e., we have

$$c_j \geq c_i + P_{ij}^{qc} \tag{6.69}$$

which follows directly from equations (6.9) of the original formulation. Combining (6.67) and (6.69) gives then the following inequality:

$$c_j - c_i \geq P_{ij}^{qc} + y_{ij}(s_{ij}^{agv} + p_j^{agv} - P_{ij}^{qc}) \tag{6.70}$$

Next, we can generalize this inequality in the following way:

Theorem 6.12 *Given AGV tasks $i, \ldots, j$, such that $i \prec \ldots \prec j$ on the same QC, the following inequalities are valid:*

$$c_j \geq c_i + P_{ij}^{qc} + \sum_{\substack{k \succeq i \\ k \prec j}} y_{kj}(P_{ik}^{qc} + s_{kj}^{agv} + p_j^{agv} - P_{ij}^{qc}) \tag{6.71}$$

$$c_j \geq c_i + P_{ij}^{qc} + \sum_{\substack{k \succ i \\ k \preceq j}} y_{ik}(s_{ik}^{agv} + p_k^{agv} + P_{kj}^{qc} - P_{ij}^{qc}) \tag{6.72}$$

Proof: We will only prove (6.71), the proof of (6.72) is similar. Clearly, because of the degree constraint on j, we have:

$$\sum_{\substack{k \succeq i \\ k \prec j}} y_{kj} \leq 1 \tag{6.73}$$

Hence, in any integer solution at most one variable is set to 1. W.l.o.g. assume $y_{kj} = 1$. Consequently, (6.71) reduces to:

$$c_j \geq c_i + P_{ik}^{qc} + s_{kj}^{agv} + p_j^{agv} \tag{6.74}$$

Obviously, we have that

$$c_k \geq c_i + P_{ik}^{qc} \tag{6.75}$$

by equation (6.69). And by equation (6.67) we have:

$$c_j \geq c_k + s_{kj} + p_j^{agv} \tag{6.76}$$

Combining equations (6.75) and (6.76) gives then the required result. □

Clearly, in equation (6.71) we should consider only tasks k for which we have that $P_{ik}^{qc} + s_{kj}^{agv} + p_j^{agv} > P_{ij}^{qc}$. An equivalent condition holds for equation (6.72).

6.7.1 Extended path inequalities

In this section, we will show how to extend the path inequalities (6.71) and (6.72) by introducing additional variables. These additional variables represent paths between i and j that use other chains. In the remainder of this subsection, we will only extend inequality (6.71). Obviously, similar extensions are possible for inequality (6.72).

For the remainder of this subsection, define variable γ_{ij} as:

$$\gamma_{ij} = \sum_{\substack{k \succeq i \\ k \prec j}} y_{kj}(P_{ik}^{qc} + s_{kj}^{agv} + p_j^{agv} - P_{ij}^{qc}) \tag{6.77}$$

Then we can extend inequality (6.71) as follows:

Theorem 6.13 *Given tasks i, j and k, l belonging to different chains, such that $i \prec j$ and $k \prec l$, the following inequality is valid:*

$$c_j \geq c_i + P_{ij}^{qc} + \gamma_{ij} + (y_{ik} + y_{lj} - 1)(s_{ik}^{agv} + p_k^{agv} + P_{kl}^{qc} + s_{lj}^{agv} + p_j^{agv} - P_{ij}^{qc}) \tag{6.78}$$

Proof: Obviously, we only have to prove the case in which $y_{ik} + y_{lj} > 1$, otherwise inequality (6.78) is weaker than inequality (6.71).

In case $y_{ik} + y_{lj} > 1$, we must have (in any feasible solution) $y_{ik} = y_{lj} = 1$. This results in $\gamma_{ij} = 0$ by the degree constraint on j. So, as a result of $y_{ik} = y_{lj} = 1$, inequality (6.78) reduces to:

$$c_j \geq c_i + s_{ik}^{agv} + p_k^{agv} + P_{kl}^{qc} + s_{lj}^{agv} + p_j^{agv} \tag{6.79}$$

which follows immediately from equations (6.7), (6.8) and (6.9) from the original MIP formulation. □

Obviously, tasks k and l should be chosen in such a way that:

$$s_{ik}^{agv} + p_k^{agv} + P_{kl}^{qc} + s_{lj}^{agv} + p_j^{agv} > P_{ij}^{qc} \tag{6.80}$$

otherwise the inequality is redundant.

Inequality (6.78) can be extended further by additional variables. For instance, we can take any variable y_{ig} such that $g \prec k$ or any variable y_{hj} such that $h \succ l$. We then get the following inequality:

$$\begin{aligned} c_j \quad &\geq \quad c_i + P_{ij}^{qc} + \gamma_{ij} \\ &+ \left(\sum_{g \preceq k} y_{ig} + \sum_{h \succeq l} y_{hj} - 1 \right) \cdot \left(s_{ik}^{agv} + p_k^{agv} + P_{kl}^{qc} + s_{lj}^{agv} + p_j^{agv} - P_{ij}^{qc} \right) \end{aligned} \tag{6.81}$$

Note that inequality (6.81) only holds if:

$$s_{ig}^{agv} + p_g^{agv} + P_{gk}^{qc} \geq s_{ik}^{agv} + p_k^{agv} \qquad \forall g \prec k \tag{6.82}$$

and

$$P_{lh}^{qc} + s_{hj}^{agv} + p_j^{agv} \geq s_{lj}^{agv} + p_j^{agv} \qquad \forall h \succ l \tag{6.83}$$

For specific layouts of the container terminal, e.g. a loop layout (see Chapter 4), this is always the case. For the more general layouts we consider in this chapter, the condition has to be checked for a specific set of tasks.

It is also possible to extend (6.78) similarly like (6.81) by considering successors of task i and predecessors of task j. In the next section we will show that it is possible to obtain a far more general form of inequality (6.81) by taking any path between tasks i and j.

6.7.2 General path inequalities

Recall Subsection 6.5.2 in which we introduced a general cycle inequality involving both variables related to AGV and ASC tasks. In a similar way, we can describe a path from AGV task i to AGV task j. Hence, it is possible to extend inequality (6.71) using any path between tasks i and j.

Note that each variable x_{ij} or y_{ij} or parameter z_{ij} that is chosen in the path, can be associated with a certain amount of handling time, say p_{ij}. For instance, setting $x_{ij} = 1$ gives a difference in completion time between ASC task i and ASC task j of p_j^{asc} (see equation (6.6)). Similarly, we can also associate a handling time with y_{ij} and z_{ij}. We can then make the following statement:

Theorem 6.14 *Consider a set of containers $K := \{1 \dots k\}$. Let 1 and k represent containers belonging to the same chain such that $1 \prec k$. Moreover, let S_1, S_2, S_3, S_4 and S_5 be a partition of K with the following properties:*

1. *If $i \in S_2 \cup S_4 \cup S_5$ then $i + 1 \notin S_1 \cup S_2$.*

2. *Containers* $1, k-1 \in S_3 \cup S_4$

Then the following inequality is valid:

$$\begin{aligned} c_k \quad \geq \quad & c_1 + P^{qc}_{1k} + \sum_{\substack{i \succeq 1 \\ i \prec k}} y_{ik}(P^{qc}_{1i} + s^{agv}_{ik} + p^{agv}_{k} - P^{qc}_{1k}) \\ & + \left(\sum_{i_q \in S_1} x_{i_q, i_q+1} + \sum_{i_q \in S_2} \sum_{j \in N^-_{q+1}} x_{i_q j} + \sum_{i_q \in S_3} \sum_{j \in N^+_q} y_{j\, i_q+1} \right. \\ & \left. + \sum_{i_q \in S_4} \sum_{j \in N^-_{q+1}} y_{i_q j} + \sum_{i_q \in S_5} z_{i_q\, i_q+1} - (|K| - 2) \right) \cdot \left(\sum_{i \in K} p_{i\, i+1} - P^{qc}_{1k} \right) \end{aligned} \tag{6.84}$$

Proof: The proof mainly follows the same line as the proof of Theorem 6.13. Again we only have to consider the case in which the variables on the path from 1 to k have all value 1. In any other case the inequality is dominated by (6.71). As a result, we have that:

$$\sum_{\substack{i \succeq 1 \\ i \prec k}} y_{ik} = 0 \tag{6.85}$$

Obviously, the inequality then follows directly from relating a variable x_{ij} or y_{ij} or a parameter z_{ij} which is set to 1, with the corresponding handling time p_{ij}. □

Inequality (6.84) is fairly complicated. Hence, given a fractional LP-solution, deciding whether or not the LP-solution satisfies all inequalities of this class might be quite complicated. In Section 6.8 we will give a separation algorithm which only finds inequalities of a subclass of (6.84).

6.8 Separation

Given the MIP formulation of the problem as given in Section 6.3, we have discussed several classes of valid inequalities in the previous sections (6.4 to 6.7). Whenever we consider a certain class of inequalities, it is possible to include all the inequalities belonging to this class in our initial LP, that is, to include all inequalities in the root node of the Branch & Bound algorithm. However, some of these classes are very large. So, including all the inequalities would give a huge number of rows in our LP, which results in large computation times for the optimal LP in every node of the branching tree. Therefore, to keep the size of the LP small, we would like to include

only the inequalities that are violated. In order to identify violated inequalities, we have to solve the separation problem. So, given an optimal solution $\bar{x}$ to the current LP problem, we have to verify that $\bar{x}$ satisfies all inequalities of a class which we have shown to be valid for our optimization problem, or if not, find an inequality of this class that is violated by $\bar{x}$.

In this section, we will give separation algorithms for the various classes of inequalities we derived in Sections 6.4 to 6.7. Some of these procedures are exact, that is, the procedure either finds an inequality that is violated, or returns that none of the valid inequalities in the class is violated. Other procedures are heuristic, which implies that whenever no violated inequality is returned, this does not necessarily mean that there is no violated inequality.

In the following subsections, we will discuss separation procedures for various classes of inequalities, starting with the cycle inequalities.

6.8.1 Separation of extended cycle inequalities

In this section we will give an exact separation algorithm for finding (violated) extended cycle inequalities. The algorithm is based on finding shortest paths in a graph and runs in polynomial time.

Recall from Section 6.5.2 the cycle inequalities, given by equation (6.22). Obviously, this is the most general form we derived, i.e., all other cycle inequalities are within this class:

$$\sum_{i_q \in S_1} x_{i_q,i_q+1} + \sum_{i_q \in S_2} \sum_{j \in N^-_{q+1}} x_{i_q,j} + \sum_{i_q \in S_3} \sum_{j \in N^+_q} y_{j,i_q+1} + \sum_{i_q \in S_4} \sum_{j \in N^-_{q+1}} y_{i_q,j} + \sum_{i_q \in S_5} z_{i_q,i_q+1} \leq |K| - 1 \quad (6.86)$$

Where S_1, S_2, S_3, S_4, S_5 is a partition of the set $K := \{1 \dots k\}$ with $k+1 = 1$, which satisfies several conditions (see Theorem 6.5). Next, we will describe the separation algorithm for the cycle inequalities.

An exact separation procedure

Given a fractional LP solution $(\bar{x}, \bar{y})$, we now can find a violated inequality (6.22) as follows. Construct a directed graph which node set is given by the set of AGV and ASC tasks. The arc set and the arc lengths are defined as follows:

1. For each pair of AGV tasks i, j we define an arc (i, j) with length:

$$1 - \max \left\{ \sum_{k \in N_i^+} \bar{y}_{k\,j}, \sum_{k \in N_j^-} \bar{y}_{ik} \right\} \tag{6.87}$$

2. For each pair of ASC tasks i', j' related to the same ASC, we define an arc (i', j') of length $1 - \bar{x}_{i'j'}$.

3. For each combination of an AGV task i and an ASC task j', we define an arc (i, j') of length

$$1 - \sum_{k \in N_i^+} \bar{y}_{k\,j} \tag{6.88}$$

where j is the corresponding AGV task of ASC task j'. Note that these arcs correspond to containers i, j in the cycle inequality (6.86) such that $i \in S_3$ and $j \in S_1 \cup S_2$.

4. For each combination of an ASC task i' and an AGV task j, we take an arc (i', j) of length

$$1 - \sum_{k \in N_j^-} \bar{x}_{i'k'} \tag{6.89}$$

where k' is the ASC task that corresponds to AGV task k. These arcs represent containers i, j in the cycle such that $i \in S_2$ and $j \in S_3 \cup S_4 \cup S_5$.

5. For any pair of AGV tasks i, j such that $i \prec j$, we take an arc (i, j) of length 0.

Note that for a pair of ASC and AGV tasks related to the same container, we do not introduce an arc. The connection between these tasks is implicitly made by the arcs in 3. and 4.

Given the graph as described above, we then calculate the shortest path from each node to itself. If the length of any of these paths is less than one, we have found a violated extended cycle inequality. Note that we can determine which arcs are in the cycle inequality by backtracking the shortest path.

Since calculating a shortest path can be done in polynomial time and since we have to do this for each node n, we may use the Floyd-Warshall algorithm for calculating all pairs shortest paths. This algorithm requires $\mathcal{O}(n^3)$ time.

Some modifications may be possible to improve the calculation of the shortest paths. Note that we do not have to consider variables with value zero, since these can never be in a violated cycle inequality. Thus we do not have to add arcs to our graph that are related to variables which value equals zero. Therefore, the graph is sparser, which increases the speed of the shortest path algorithm.

6.8.2 Separation of clique inequalities

In this subsection, we will discuss the separation problem of the clique inequalities as given by equation (6.15). The procedure we propose is heuristic and can be applied to the separation of cut set inequalities as well (see also Subsection 6.8.3).

Consider a mixed clique inequality given by equation (6.15). Such a clique inequality is violated if we have that:

$$\sum_{i \in C_1} \sum_{j \in K} x_{ji} + \sum_{i \in C_2} \sum_{j \in K} y_{ji} > |K| - 1 \tag{6.90}$$

For the remainder of this subsection, we will assume that subset C_1 is empty, i.e., we have a clique of AGV tasks only. Clearly, the results can be easily extended to the case in which we have a clique of both AGV and ASC tasks.

Let $\bar{y}_{ij}$ denote a fractional solution, obtained from the LP-relaxation. Now, we want to find a set of tasks K such that the following term is maximal:

$$\sum_{i,j \in K} \bar{y}_{ij} - |K| + 1 \tag{6.91}$$

That is, we look for a clique K for which the corresponding clique inequality is most violated.

Next, we will consider some algorithmic aspects of finding a violated clique inequality. First, we will give a MIP formulation of the problem, followed by a heuristic.

A MIP formulation

The problem of finding the most violated clique inequality, given a fractional LP solution $\bar{y}$, can be formulated as a MIP. Therefore, we introduce the following variables:

$$\alpha_i = \begin{cases} 1 & \text{if task } i \text{ is chosen in the clique} \\ 0 & \text{otherwise} \end{cases}$$

$$\beta_{ij} = \begin{cases} 1 & \text{if variables } \bar{y}_{ij} \text{ and } \bar{y}_{ji} \text{ are chosen in the clique} \\ 0 & \text{otherwise} \end{cases}$$

In order to find the most violated clique inequality, we can formulate the following optimization problem:

$$\text{Min} \quad \eta = \sum_{i \in N} \alpha_i \; - \sum_{\substack{i,j \in N \\ j>i}} \beta_{ij}(\bar{y}_{ij} + \bar{y}_{ji}) \tag{6.92}$$

s.t.

$$\sum_{i \in N} \alpha_i \geq 2 \tag{6.93}$$

$$\beta_{ij} \leq \alpha_i \qquad \forall\, i,j \in N : j > i \tag{6.94}$$

$$\beta_{ij} \leq \alpha_j \qquad \forall\, i,j \in N : j > i \tag{6.95}$$

$$\beta_{ij} \geq 0 \qquad \forall\, i,j \in N : j > i \tag{6.96}$$

$$\beta_{ij} \leq 1 \qquad \forall\, i,j \in N : j > i \tag{6.97}$$

$$\alpha_i \in \{0,1\} \qquad \forall\, i \in N \tag{6.98}$$

The objective (6.92) is to find a clique of maximal weight. Whenever the value of the objective function $\eta < 1$, we have a violated clique inequality. Whenever no task is selected into the clique, the objective function is minimal. In order to avoid this, we introduce restriction (6.93) which states that a clique should at least contain two or more tasks. Equations (6.94) and (6.95) state that β_{ij} can only be set to 1, whenever both task i and task j are selected in the clique. That is, $\beta_{ij} = 1$ implies $\alpha_i = 1$ and $\alpha_j = 1$. Note that β_{ij} appears with a negative coefficient in the objective. So, β_{ij} will always be set as large as possible. As a result, we do not have to define β_{ij} as a binary variable. Instead, we define β_{ij} as a continuous variable between 0 and 1, i.e., equations (6.96) and (6.97). Finally, equation (6.98) requires α_i to be either 0 or 1.

The problem of finding a maximal edge-weighted clique, as formulated by (6.92) to (6.98) is NP-complete, cf. Dijkhuizen and Faigle (1993), since it generalizes the maximal clique problem. Therefore, we will use a heuristic approach for finding a violated clique inequality. This heuristic is described in the following subsection.

A heuristic

In this subsection, we will describe an algorithm for finding a maximal edge-weighted clique of size K. For convenience, we assume an undirected graph, where w_e is the weight associated with an edge e between two nodes i, j. Hence, for a LP solution $\bar{y}$, we define $w_e := y_{ij} + y_{ji}$.

The algorithm that finds a maximal edge weighted clique of size K simply starts by choosing randomly a clique of size K. Given this initial clique, we then try to

exchange a node c that is in the clique with a node d that is not in the clique. If the sum of the edge weights increases, we replace node c by node d, else not. This local search algorithm looks then as follows:

Algorithm 1 Separation clique inequality of size K

```
Given a graph G(N, E) with edge weights w_e
Select randomly a clique C of size K
while improvement do
  for all c ∈ C do
    for all d ∉ C do
      C' := (C \ {c}) ∪ {d}
      if Σ_{e∈C'} w_e > Σ_{e∈C} w_e then
        C := C'
      end if
    end for
  end for
end while
```

Obviously, it is possible to run the algorithm several times in a row, that is, to start with different randomly chosen initial cliques. If for the clique C that is found by the algorithm we have that $\sum_{e \in C} w_e > K - 1$, we have found a violated clique inequality on C.

6.8.3 Separation of cut set inequalities

In this section, we will discuss the separation of cut set inequalities. We will show that the problem of finding a violated cut set inequality can be transformed into finding a violated edge weighted clique inequality. As a result, finding a violated cut set inequality can be done using the same method as described in Section 6.8.2.

Recall from Section 6.5.3 the cut set inequalities:

$$\sum_{\substack{q \in Q \\ q \neq 1}} \sum_{j \in N_q^+} y_{j,i_1} + \ldots + \sum_{\substack{q \in Q \\ q \neq |Q|}} \sum_{j \in N_q^+} y_{j,i_{|Q|}} \leq |C| - 1 \tag{6.99}$$

$$\sum_{\substack{q \in Q \\ q \neq 1}} \sum_{j \in N_q^-} y_{i_1,j} + \ldots + \sum_{\substack{q \in Q \\ q \neq |Q|}} \sum_{j \in N_q^-} y_{i_{|Q|},j} \leq |C| - 1 \tag{6.100}$$

From the definition of the cut sets, it follows that we have at most $\mathcal{O}(n^{|QC|})$ different cut sets, where n denotes the total number of containers and $|QC|$ is the total number of QCs (chains). Clearly, it would be possible to enumerate all different cut set inequalities, since for a fixed number of QCs, we have a polynomial number of

them. However, for $|QC|$ in the range 4 to 6 and for large n, the required computation time may still be too large. Hence, we transform the problem of finding a violated cut set inequality into finding a maximal edge weighted clique in a graph. We can then solve the separation problem using the heuristic algorithm for the clique inequalities as given in Subsection 6.8.2. It is obvious that since we solve the clique problem heuristically, we also cannot guarantee that a violated cut set inequality is always found.

Now we will show how to transform the cut set inequalities into a graph with corresponding edge weights. For each AGV task, we introduce a node in the graph. Note that by definition of a cut set, we only want to select at most one node from every chain in the cut set. Therefore, we define only edges between nodes of different chains. In case of a cut set inequality given by (6.99), the edge-weight for an edge e between nodes i and j is then calculated as follows:

$$w_e := \sum_{k \succeq i} y_{kj} + \sum_{k \succeq j} y_{ki} \tag{6.101}$$

In case we want to search a violated cut set inequality given by (6.100), the edge-weights are defined as:

$$w_e := \sum_{k \preceq j} y_{ik} + \sum_{k \preceq i} y_{jk} \tag{6.102}$$

Now, given the weights w_e, we want to select a clique C such that

$$\sum_{e \in C} w_e > |C| - 1 \tag{6.103}$$

In case we find such a clique C, we have found a violated cut set inequality. Thus, we have transformed the problem of finding a violated cut set inequality into the problem of finding a violated clique inequality on the graph as defined above.

6.8.4 Separation of inequalities on completion times of tasks

In this subsection, we will discuss the separation of the inequalities as given in Section 6.6. We will only discuss the separation for inequalities (6.54), the separation for inequalities (6.58) can be done in a similar way.

Recall inequality (6.54) from Section 6.6:

$$\sum_{i \in S} p_i b_i \geq \sum_{i \in S} p_i^2 + \sum_{i \in S} \sum_{\substack{j \in S \\ j > i}} p_i p_j \tag{6.104}$$

Clearly, we would like to find the most violated inequality, that is, we would like to maximize the right hand side minus the left hand side:

$$\max_{S \subseteq N} \left\{ \sum_{i \in S} p_i^2 + \sum_{i \in S} \sum_{\substack{j \in S \\ j > i}} p_i p_j - \sum_{i \in S} p_i b_i \right\} \tag{6.105}$$

Obviously, it is possible to enumerate over all possible sets S in order to find the most violated inequality. However, as $|S|$ increases, the number of possible sets explodes. An algorithm proposed by Queyranne (1993), requires only $\mathcal{O}(n)$ evaluations. The algorithm is based on the following observation (for details see Queyranne (1993)):

Property 6.8.1 *Let S be the set that maximizes equation (6.105), given the relaxation $\bar{b}$. Then for each pair of tasks i, j such that $\bar{b}_j < \bar{b}_i$, we have that:*

$$i \in S \Rightarrow j \in S \tag{6.106}$$

For given $\bar{b}_i$'s, Property 6.8.1 implies that out of all candidate sets S, we only have to consider $|N|$ of them, namely, the ones which contain the n^{th} smallest $\bar{b}_i$'s, for each $n := 1 \ldots |N|$. Algorithm 2 sketches the outline of the separation procedure.

Algorithm 2 Separation inequality on completion times (Queyranne (1993))

Given tasks {1, ... , n}, sorted in non-decreasing order of the $\bar{b}_i$'s.
W.l.o.g. assume that $\bar{b}_1 \leq \ldots \leq \bar{b}_n$.
$S = \emptyset$; $S_{best} = \emptyset$; $best = 0$;
for i := 1 to n **do**
  $S := S \cup \{i\}$
  if $best < \sum_{i \in S} p_i^2 + \sum_{i \in S} \sum_{j \in S, j > i} p_i p_j - \sum_{i \in S} p_i \bar{b}_i$ **then**
    $best = \sum_{i \in S} p_i^2 + \sum_{i \in S} \sum_{j \in S, j > i} p_i p_j - \sum_{i \in S} p_i \bar{b}_i$
    $S_{best} := S$
  end if
end for

Starting with an empty set S and adding the tasks one by one, the actual calculations of (6.105) can be done recursively. This allows for a very efficient implementation of the algorithm. The algorithm is exact, that is, it either gives us the most violated inequality, or it returns that there is no inequality violated.

For the case of inequality (6.55), a similar algorithm can be constructed. Also whenever we introduce tails, i.e., equation (6.58), we can still use the same algorithm in a somewhat modified version. However, in this case, the separation is not exact

anymore. That is, Property 6.8.1 does not necessarily hold for the set S which represents the most violated inequality. As a result, there might be a set S' for which the inequality is violated, but this set S' might not belong to the group of $|N|$ sets that are considered by the algorithm. Therefore, the algorithm is used as a heuristic procedure for separating inequalities (6.58).

6.8.5 Separation of path inequalities

In this section we will give a separation procedure for the inequalities which were discussed in Section 6.7. Recall the basic path inequalities:

$$c_j \geq c_i + P^{qc}_{ij} + \sum_{\substack{k \succeq i \\ k \prec j}} y_{kj}(P^{qc}_{ik} + s^{agv}_{kj} + p^{agv}_{j} - P^{qc}_{ij}) \tag{6.107}$$

$$c_j \geq c_i + P^{qc}_{ij} + \sum_{\substack{k \succ i \\ k \preceq j}} y_{ik}(s^{agv}_{ik} + p^{agv}_{k} + P^{qc}_{kj} - P^{qc}_{ij}) \tag{6.108}$$

Note that in both path inequalities, we will only include variables y_{kj} for which we have that:

$$s^{agv}_{kj} + p^{agv}_{j} > P^{qc}_{kj} \tag{6.109}$$

Obviously, checking whether or not a path inequality is violated, can be done by enumerating over all i, j in the same chain. This requires at most $\mathcal{O}(n^2)$ evaluations.

In order to determine whether or not there is an general path inequality violated, i.e., an inequality like (6.84), we propose the following procedure. For each chain separately, we will search for a violated inequality. Note that we are only interested in path inequalities for which the values of the binary variables that represent the path are sufficiently large. Therefore, we construct a graph in which we will search a shortest path (consisting of binary variables) from AGV task i to AGV task j in the same chain.

The graph is then constructed as follows. For each task we take a node. The arc set is defined in a similar way as it is in the graph we use for the separation of the cycle inequalities (see Section 6.8.1). The only modification is that between nodes belonging to the current chain under consideration, we take no arcs. Moreover, from nodes in the current chain to any other node, we only take arcs representing variables y_{ij} . This follows directly from the definition of the generalized path inequalities (see Theorem 6.14).

Next, we calculate a shortest path in the graph from node i to node j, where i, j are in the current chain. If the length of this shortest path is greater or equal to

one, we may conclude that there is no violated path inequality on nodes i and j. If the length of the path is less than one, this path may induce a violated inequality. We then have to check this by taking the basic path inequality for tasks i and j and adding the variables representing the path, multiplied with their coefficients p_{ij} (see also Subsection 6.7.2).

Note that the procedure sketched above does not give certainty about the existence of a violated inequality. Suppose that there are two paths between nodes i and j in the constructed graph, both with length less than 1. The shortest path algorithm will find the shortest of the two paths. The corresponding path inequality may not be violated, while the inequality corresponding to the other path may.

An alternative procedure to find a violated extended path inequality, i.e., inequality (6.78), is the following. Check for which pair of AGV tasks i, j the difference in completion times $c_j - c_i$ is tight, that is, for which i, j we have:

$$c_j - c_i \approx P_{ij}^{qc} \tag{6.110}$$

For such a pair of tasks, there might be a violation of the path inequality (6.78). Next, look for a violation by considering only paths through tasks k, l such that $k \prec l$. In this case, we simply enumerate of all possible k, l.

6.9 Preprocessing and logical constraints

In this section we will derive several preprocessing rules and logical constraints that can be applied in any node of the Branch & Bound tree. These rules allow us to set variables equal to their upper or lower bound. Moreover, it is possible to introduce additional restrictions which relate the values of several variables to each other. In order to justify these rules, we have to show that by applying them it is not possible that the entire set of optimal solutions is eliminated from the search tree. Clearly, this would be the case if solutions are only eliminated from consideration when it has been established that there exists a solution with a strictly better objective value. However, our rules do not have this property, since we can only show that there exists another solution in the tree with an objective value that is not worse. Therefore, we will use a more subtle argument, for which we introduce the following three additional criteria:

1. The number of pairs of tasks i, j for which i is scheduled before j on the ASC, but for the corresponding containers i, j we have that $j \prec i$ on the QC (to be minimized).

2. The number of pairs of tasks i, j for which i is scheduled before j on the ASC such that $p_i^{asc} > p_j^{asc}$ (to be maximized).

3. The variance in waiting time of AGVs between the completion of the previous task and the start of the next task, i.e., $\sum_{i,j}((b_j - s_{ij}^{agv} - c_i)y_{ij})^2$ (to be minimized).

The objective value and the three additional criteria (in the given order) define a lexicographical ordering on the solutions. For each preprocessing rule, we will show that certain solutions can be eliminated from consideration because, somewhere else in the tree, there exists a strictly better solution with respect to this ordering.

Theorem 6.15 *If solutions are only eliminated when there exists a better solution with respect to the lexicographical ordering defined by the objective value and the additional criteria 1., 2. and 3., then this will never lead to the deletion of the entire set of optimal solutions, i.e., there always remains at least one optimal solution in the tree.*

Proof: Consider the set of optimal solutions. By definition, these solutions all have the same objective value. Take the subset of optimal solutions for which the value of criterion 1 is lowest. From this subset, take the subset of all solutions with highest value of criterion 2 and, finally, take from that set the subset of solutions with the lowest value of criterion 3. The optimal solutions in the latter subset are "the best" with respect to the lexicographical ordering and, therefore, can never be eliminated. □

6.9.1 Preprocessing on x_{ij} variables

In this section, we will discuss some preprocessing rules on the x_{ij} variables which represent the order in which the tasks are scheduled on an ASC. Our first result is the following.

Theorem 6.16 *For all containers i and j such that $j \prec i$ on a QC and $p_j^{asc} \geq p_i^{asc}$, we have $x_{ij} = 0$.*

Proof: Suppose $x_{ij} = 1$. We will show that setting $x_{ji} = 1$ instead, gives a schedule with an objective value that is as least as good, but which is better with respect to criterion 1.

It is obvious that setting $x_{ji} = 1$ instead of $x_{ij} = 1$ gives us a better value for criterion 1. Hence, the only thing we have to show is that we get a schedule which

has an objective value that is as least as good. We will do this by showing that setting $x_{ji} = 1$ instead of $x_{ij} = 1$, has no effect on the total time the ASC requires to handle both containers. Moreover, we will show that the completion times of the corresponding AGV tasks do not increase. Hence, it is possible to complete both tasks within the time of the original schedule.

Assume $x_{ij} = 1$ and let $t_1, t_2, t_1 \leq t_2$ denote the arrival times of the AGVs that transport containers i and j. Note that the containers are not transported by the same AGV, since $y_{ij} = 1$ would contradict $j \prec i$. Moreover, $p_j^{asc} \geq p_i^{asc}$, so reversing the tasks will not lead to additional waiting time of the ASC for the AGVs. This can be shown as follows. In case of $x_{ij} = 1$, we have by definition that $t_1 \leq b_i$ and $t_2 \leq b_j$. Furthermore, $b_j \geq b_i + p_j^{asc}$. In case of setting $x_{ji} = 1$, we get the following completion times on the ASC:

$$b_j^{new} = b_i + (p_j^{asc} - p_i^{asc}) \tag{6.111}$$

$$b_i^{new} = \max\{t_2,\ b_i + (p_j^{asc} - p_i^{asc}) + p_i^{asc}\} \tag{6.112}$$

It is obvious that $b_i^{new} \leq b_j$, i.e., the completion time of the last task does not increase.

Next, we complete the proof by showing that the completion times of tasks i and j on the AGV will not increase. In case of $x_{ij} = 1$, the earliest possible completion time of task j on the AGV equals $b_j + p_j^{agv}$. Since $j \prec i$, the QC can start handling the containers not earlier than $b_j + p_j^{agv}$. For the case of $x_{ji} = 1$ it is now sufficient to show that both containers are also available at the QC at latest at time $b_j + p_j^{agv}$. If this is the case, the completion times of the tasks on the AGVs will not be larger than in the case of $x_{ij} = 1$. Since $b_i^{new} \leq b_j$, and $p_i^{agv} = p_j^{agv}$, we have that both AGVs are available at the QC at time less than or equal to $b_j + p_j^{agv}$, which completes the proof. □

Another inequality that we can derive whenever $x_{ij} = 1$, is the following.

Theorem 6.17 *Suppose $x_{ij} = 1$ for containers i and j such that $j \prec i$ on a QC. Then we can assume:*

$$b_j < b_i + p_i^{asc} \tag{6.113}$$

Proof: Note that in case $p_j^{asc} \geq p_i^{asc}$ we cannot have $x_{ij} = 1$ because of Theorem 6.16. Hence we have $p_j^{asc} < p_i^{asc}$. Now assume that (6.113) does not hold. As in the proof of Theorem 6.16, we will show that reversing tasks i and j will not lead to a schedule that is worse, while setting $x_{ji} = 1$ instead of $x_{ij} = 1$ leads to a better schedule with respect to criterion 1.

Reversing the order in which tasks i and j are scheduled does not increase the time window compared to the situation in which $x_{ij} = 1$. Since $p_j^{asc} < p_i^{asc}$, task j can be completed before time b_i. Moreover, since equation (6.113) does not hold, there is sufficient time to schedule task i before time b_j.

Since we showed that the completion time of task i on the ASC is less than or equal to the completion time of task j on the ASC in case of $x_{ij} = 1$, the remainder of the proof is similar to the proof of Theorem 6.16. □

Theorem 6.18 *Suppose $x_{ij} = 1$ for containers i and j such that $j \prec i$ on a QC. Moreover, suppose $p_i^{asc} - p_j^{asc} \leq p_j^{qc}$. Then for k such that $x_{jk} = 1$ we may assume:*

$$b_k < b_i + p_i^{asc} + p_k^{asc} \tag{6.114}$$

Proof: Again we have that $p_i^{asc} > p_j^{asc}$ by Theorem 6.16. We will show that by reversing tasks i and j on the ASC, we obtain a schedule which has an objective value that is as least as good, and that is better with respect to criterion 1.

It is obvious that if (6.114) does not hold, setting $x_{ji} = x_{ik} = 1$ instead of $x_{ij} = x_{jk} = 1$ does not lead to a larger time window required to handle containers i, j and k on the ASC. Moreover, note that the completion time of task k on the ASC, i.e., b_k does not increase.

By Theorem 6.17 we have that $b_j < b_i + p_i^{asc}$. Hence, reversing the order of i and j, leads in the worst case to a completion time of task i on the ASC which is larger than the completion time of task j on the ASC in case $x_{ij} = 1$. So, as a consequence of $x_{ji} = 1$, we have:

$$b_j \leq b_i^{new} \leq b_j + (p_i^{asc} - p_j^{asc}) \tag{6.115}$$

Since $j \prec i$, the completion time of task i on the AGV is at least:

$$c_{\geq} b_j + p_j^{agv} + p_j^{qc} \tag{6.116}$$

whether $x_{ij} = 1$ or $x_{ji} = 1$. So, we only have to show that in case $x_{ji} = 1$, this completion time will not be larger. Given the completion time of task i on the ASC, we have that the AGV is ready at the QC for unloading (completion time of AGV task i) at time:

$$b_j + (p_i^{asc} - p_j^{asc}) + p_i^{agv} \tag{6.117}$$

which is not larger than (6.116) since $p_i^{agv} = p_j^{agv}$ and $p_i^{asc} - p_j^{asc} \leq p_j^{qc}$ by assumption. □

Theorem 6.19 *Suppose $x_{ij} = 1$ and $p_j^{asc} > p_i^{asc}$ for containers i and j. Then for the corresponding AGV tasks, we can assume:*

$$c_i < b_j + p_i^{agv} \tag{6.118}$$

Proof: Note that containers i and j are handled by different QCs, otherwise $x_{ij} = 0$ by Theorem 6.16. Now suppose (6.118) does not hold, i.e., $c_i \geq b_j + p_i^{agv}$. We will show that for any schedule for which $x_{ij} = 1$ and that does not satisfy (6.118), the alternative schedule that results after reversing i and j on the ASC has a makespan at least as good. Note that since i and j belong to different QCs, reversing the containers on the ASC cannot affect the value of criterion 1. However, it will lead to a better value of criterion 2 since $p_j^{asc} > p_i^{asc}$ and j is scheduled first if $x_{ji} = 1$.

The case in which the same AGV transports both containers i and j is trivial, i.e., $y_{ij} = 1$ implies $b_j > c_i$ which contradicts $c_i \geq b_j + p_i^{agv}$. Hence we assume that two different AGVs transport the containers.

First, we will show that reversing tasks i and j on the ASC leads to a strictly earlier arrival time of container j at the QC. Moreover, we will show that the completion times of tasks i and j on the AGV are not worse. Since task j is now scheduled first on the ASC, we have $b_j^{new} < b_j$ and thus, also the arrival time at the QC is smaller for container j. Moreover, the time window required to handle both containers j and i on the ASC does not increase since $p_j^{asc} > p_i^{asc}$ (see also Theorem 6.16), i.e., we have $b_i^{new} \leq b_j$. So, the arrival time of container i at the QC is less than or equal to $b_j + p_i^{agv}$. Since we assumed that (6.118) does not hold, we have arrived at the desired result for c_i, the AGV completion time of task i.

□

So far, we only assumed that a single variable x_{ij} is set to one. Based on this variable, we were able to derive some additional inequalities. Next, we consider the case in which we have multiple variables set to 1. More specific, we consider the case of a partial schedule on the ASC, i.e., for tasks $i, \ldots, j$ we have an ordering represented by the variables $x_{ik}, \ldots, x_{lj}$, all set to 1. Now suppose we have that $j \prec i$. Then we can make the following claim.

Theorem 6.20 *Suppose we have a partial schedule on the ASC for tasks $i, \ldots, j$, such that container j precedes container i on the QC. Then for task k such that $x_{ik} = 1$, we must have:*

$$p_i^{asc} \geq p_k^{asc} \tag{6.119}$$

Proof: Suppose (6.119) does not hold, i.e., $p_i^{asc} < p_k^{asc}$. Then by Theorem 6.19 we may assume that $c_i < b_k + p_i^{agv}$. However, since k is scheduled before j on the ASC,

i.e., $b_k < b_j$, we have $c_i < b_j + p_i^{agv}$. Moreover, since i and j are handled by the same QC, and thus $p_i^{agv} = p_j^{agv}$, this would lead to $c_i < b_j + p_i^{agv} \leq c_j$. Clearly this cannot be the case because $j \prec i$ implies $c_j < c_i$. □

Theorem 6.21 *Given a partial schedule on the ASC for tasks $i, \ldots, j$, such that container j precedes container i on the QC. Let l be the task scheduled directly after task i. Then for all other tasks k scheduled between i and j (including j) on the ASC, such that $p_k^{asc} > p_i^{asc}$ and there is no container $m \in i, \ldots, k$ such that $m \prec k$, we have that:*

$$b_l < b_i + (p_k^{asc} - p_i^{asc}) + p_l^{asc} \tag{6.120}$$

Proof: Suppose (6.120) does not hold. Reversing tasks i and k on the ASC will lead to an earlier completion time of k on the ASC. Moreover, since (6.120) does not hold, the completion time of task l on the ASC is not influenced. For i we get a larger completion time on the ASC. However, the completion time of i on the AGV will not increase. This is obvious since the completion of task i on the AGV is still blocked by task j, which is scheduled after task i on the ASC in any case.

Note that we assumed that container k does not have any predecessor that was scheduled before task k on the ASC in the original schedule. Hence, swapping tasks i and k in the schedule will not lead to a worse value with respect to criterion 1. Moreover, the schedule is better in terms of criterion 2, since $p_k^{asc} > p_i^{asc}$. □

6.9.2 Preprocessing on y_{ij} variables

Similar statements like in Subsection 6.9.1 can also be made whenever variables y_{ij} are set to 1. Consider for instance the following:

Theorem 6.22 *Suppose $y_{ki} = y_{lj} = 1$. Moreover assume container k precedes container l on a QC. Then we should have:*

$$b_i - s_{ki}^{agv} \leq b_j - s_{lj}^{agv} \tag{6.121}$$

Proof: Again we prove by contradiction, i.e., we assume (6.121) does not hold. We will show that setting $y_{kj} = y_{li} = 1$ will never give a worse schedule. Obviously, whenever i and j are handled by different ASCs, criteria 1 and 2 are not influenced. In case the containers are handled by the same ASC, and assuming (6.121) does not hold, we have $b_i > b_j$, i.e., task i is scheduled after task j. However, setting $y_{kj} = y_{li} = 1$ instead of $y_{ki} = y_{lj} = 1$ only influences the AGV assignment and

thus, does not influence criterion 1 or criterion 2. Moreover, we will show that with respect to criterion 3, setting $y_{kj} = y_{li} = 1$ gives a better solution. Since $k \prec l$, that is, containers k and l are handled by the same QC, we have that $s_{ki}^{agv} = s_{li}^{agv}$ and $s_{kj}^{agv} = s_{lj}^{agv}$. With respect to criterion 3, setting $y_{ki} = y_{lj} = 1$ gives the following:

$$(b_i - s_{ki}^{agv} - c_k)^2 + (b_j - s_{lj}^{agv} - c_l)^2 \tag{6.122}$$

However, setting $y_{kj} = y_{li} = 1$ gives:

$$(b_i - s_{ki}^{agv} - c_l)^2 + (b_j - s_{lj}^{agv} - c_k)^2 \tag{6.123}$$

which is obviously smaller than (6.122), since $b_i - s_{ki}^{agv} > b_j - s_{lj}^{agv}$ and $c_k < c_l$. □

Given Theorem 6.22, we can make the following statement.

Corollary 6.23 *Suppose we have $y_{ki} = y_{lj} = 1$. Moreover, assume $k \prec l$ and i and j to be handled by the same ASC. Then $x_{ji} = 0$.*

Proof: Setting $x_{ji} = 1$ would imply $b_i > b_j$, which clearly contradicts Theorem 6.22. □

6.9.3 Preprocessing on x_{ij} and y_{ij} variables

Whenever both x_{ij} and y_{ij} variables are set to 1, we can derive the following statement about the completion times of jobs:

Theorem 6.24 *Suppose $x_{ij} = y_{ki} = 1$ for containers i, j, k. Moreover assume $j \prec i$ on a QC. Then we may assume:*

$$c_k + s_{ki}^{agv} > b_i - p_i^{asc} + p_j^{asc} \tag{6.124}$$

Proof: Note that we may assume $p_j^{asc} < p_i^{asc}$ since $p_j^{asc} \geq p_i^{asc}$ would imply $x_{ij} = 0$ by Theorem 6.16.

Again suppose (6.124) does not hold, i.e., the AGV transporting container i arrives at the ASC before time $b_i - p_i^{asc} + p_j^{asc}$. Moreover, the ASC finishes task j by definition at time $b_j \geq b_i + p_j^{asc}$. We will show that scheduling tasks i and j in reverse order on the ASC and switching the assignment of the AGVs as well, does not lead to a worse schedule. Obviously, setting $x_{ji} = 1$ instead of $x_{ij} = 1$ gives a better value for criterion 1.

Since we assume (6.124) does not hold, scheduling task j before task i on the ASC and switching the AGV assignment will lead to the completion of task j on the

ASC at latest at time $b_i - p_i^{asc} + p_j^{asc}$. Note that at this time, the ASC can advance to container i, since the AGV has already arrived at this time. So, the ASC is ready to load container i on the AGV at time $b_i + p_j^{asc}$ which is clearly less than or equal to b_j in case of $x_{ij} = 1$, regardless of the arrival time of the AGV. □

6.10 Computational testing

In this section, we will report on some computational experiments that were carried out with the MIP formulation as given in Section 6.3 and the valid inequalities we derived for this formulation in the previous sections. We will test the effectiveness of the various classes of inequalities by adding the whole class to the MIP and then using the commercial solver CPLEX (version 6.6) to solve the model. The effectiveness of a class of inequalities can be measured by considering the number of required nodes in the Branch & Bound tree that CPLEX needs to solve the model. Next to adding a whole class of inequalities, we also consider a cutting plane approach in which we repeatedly solve the LP relaxation in the root node of the search tree and separate valid inequalities, until we cannot find any violated valid inequality. We then give the resulting problem to CPLEX to solve.

In the testing of the inequalities, we will not consider all classes, simply because some of them are (exponentially) large. Initially, we will test the effectiveness of the following inequalties:

1. The cut set inequalities given by equations (6.26) and (6.27).

2. The mixed cut set inequalities given by equation (6.48). We only consider cut sets C and C' such that for each pair $i \in C$, $i' \in C'$, we have that i' immediately succeeds i in the chain.

3. The inequalities on the completion time variables as discussed in Section 6.6, i.e., inequalities (6.54) and (6.58).

4. The path inequalities in their basic form, i.e., inequalities (6.71) and (6.72), which only consider variables related to AGV tasks of containers handled by the same QC.

5. Addition of some logical constraints. Note that for instance the constraints of Theorem 6.17 and Theorem 6.19, which result from setting binary variables

number of containers	LP value root node	optimal (best MIP)	number of nodes B & B	added inequalities
20	662	942	53660	-
20	662	942	159619	1
20	662	942	110878	2
20	942	942	61783	3
20	662	(1494)	250000	4
20	662	942	30669	5
20	662	942	9103	6

Table 6.1: Effectiveness of valid inequalities

equal to one, can be rewritten as an inequality using a big constant K:

$$b_j < b_i + p_i^{asc} + (1 - x_{ij})K \qquad \forall j \prec i \tag{6.125}$$

$$c_i < b_j + p_i^{agv} + (1 - x_{ij})K \qquad \forall i, j : p_j^{asc} > p_i^{asc} \tag{6.126}$$

Moreover, we set variables equal to zero according to Theorem 6.16. Other logical constraints are not considered.

6. A cutting plane approach in which we repeatedly solve the LP relaxation and the separation problems in the root node, until no violated inequality can be found. So, we add only the violated inequalities that are found by the separation procedures instead of adding whole classes of inequalities. We consider all the inequalities for which we discussed a separation procedure in Section 6.8. After the addition of the violated inequalities in the root, we solve the problem using CPLEX.

As a benchmark, to compare the effectiveness of the various inequalities, we take the number of nodes in the Branch & Bound tree that CPLEX requires to solve the MIP formulation given by equations (6.1) to (6.12), without any additional valid inequalities. Since in some cases, CPLEX does not manage to solve the problem to optimality in reasonable time, we set the limit of the number of nodes in the Branch & Bound tree to $2.5 \cdot 10^5$. In Tables 6.1 to 6.4, we give computational results for some (small) but typical instances of general layouts. Again, all instances were generated using the simulation model as discussed in Section 3.3. In case the optimal solution is not found within $2.5 \cdot 10^5$ nodes, we give the best solution found so far (in brackets).

From Tables 6.1 to 6.4 we conclude the following. The inequalities on the binary variables only (1,2), have a variable effect on the required number of nodes in the

number of containers	LP value root node	optimal (best MIP)	number of nodes B & B	added inequalities
20	738	1034	238426	-
20	738	1034	165302	1
20	738	1034	74201	2
20	1034	1034	17578	3
20	738	1034	62889	4
20	738	1034	20360	5
20	1034	1034	1593	6

Table 6.2: Effectiveness of valid inequalities

number of containers	LP value root node	optimal (best MIP)	number of nodes B & B	added inequalities
20	928	1170	27953	-
20	928	1170	32663	1
20	928	(1315)	250000	2
20	1170	1170	50418	3
20	928	1170	18658	4
20	928	(1261)	250000	5
20	1170	1170	6916	6

Table 6.3: Effectiveness of valid inequalities

number of containers	LP value root node	optimal (best MIP)	number of nodes B & B	added inequalities
20	881	(921)	250000	-
20	881	(903)	250000	1
20	881	(954)	250000	2
20	881	903	2917	3
20	881	903	116750	4
20	881	903	3225	5
20	903	903	280	6

Table 6.4: Effectiveness of valid inequalities

Branch & Bound tree. In one case they are very effective (Table 6.2), in the other cases not. Moreover, the size of these classes is quite large, that is, there are many inequalities of this type. Adding all these inequalities therefore results in a large LP, which again results in slower computation times. Note that the results in Tables 6.1 to 6.4 only consider the number of nodes in the search tree and not the required computation time.

The inequalities on the continuous variables (3) have a large impact on the value of the LP relaxation in the root of the search tree. In most cases, this LP value turns out to be equal to the value of the optimal solution. Moreover, the number of nodes that are needed to find the optimal solution, is decreased considerably.

For the path inequalities (4) and the logical constraints (5), the results vary. In some cases they perform (very) well, in other cases they perform even worse than the MIP without any added inequalities.

Finally, solving the problem by adding only *violated* inequalities in the root node of the tree seems the best option. This not only results in a good value of the LP relaxation in the root of the tree (in all cases, this value is equal to the value of the optimal solution), it also reduces the number of nodes required in the Branch & Bound tree significantly. For instance, in Tables 6.2 and 6.4, the number of nodes is only a fraction of the number of nodes that is required without adding any cuts.

Hence, we may conclude that although most classes of inequalities we derived are quite effective, solving the problem with a cutting plane approach in the root of the search tree gives the best results. In Table 6.5 we give some additional computational evidence on this. In this table, we not only compare the number of required nodes in the Branch & Bound tree that CPLEX needs, we also compare the computation times. Next to the total required time of the cutting plane approach, we also give the time required in the cutting phase, i.e., the time required to repeatedly solve the separation problems and to reoptimize the LP with the added inequalities. Moreover, we do not restrict anymore the maximal number of nodes in the Branch & Bound tree to $2.5 \cdot 10^5$. Instead, we solve every instance to optimality, regardless of the computational effort.

		MIP, no valid inequalities			cutting plane approach			
number of containers	optimal solution	LP value root	number of nodes B & B	time (sec)	LP value root	number of nodes B & B	total time (sec)	time cuts (sec)
20	1034	738	238426	1353	1034	1593	96	70
20	942	662	53660	303	942	9103	126	41
20	912	881	440106	29687	903	42055	5160	149
20	1170	928	27953	201	1170	6916	162	90
20	903	881	5189684	33559	903	280	132	122
20	902	877	54975	436	902	2506	261	210
20	973	623	102559	624	973	11660	211	91

Table 6.5: Computational results for the cutting plane algorithm

As we can see from Table 6.5, the decrease in the number of nodes required by CPLEX to solve the problem after addition of violated inequalities in the root decreases significantly. Also the required computation time is far less, although a significant amount of computation time of the cutting plane approach is needed for the separation of the violated inequalities. Hence, we may conclude that the various inequalities are already very effective, even if we only add them in the root node of the search tree. Obviously, one may solve the separation problems in every node of the search tree, using a Branch & Cut approach. This will likely lead to even better results, both in the required number of nodes and the computation times. However, the lack of a commercial package that supports such an approach prevents us from applying this technique. Some computational experiments were done with the academic package ABACUS (see Thienel (1995)), but this package requires too much PC memory and can only be used to solve very small test instances.

6.11 Conclusions

In this chapter, we have considered the integrated scheduling of ASCs, AGVs and QCs at a container terminal with a general layout of the AGV area. As a result, of this general layout of the AGV area, we cannot apply the model as presented in Chapter 4.

We have formulated a Mixed Integer Programming (MIP) formulation for this scheduling problem. Moreover, we derived several classes of valid inequalities. Some of these classes consider only the binary variables of our MIP formulation, some only the continuous variables and some inequalities combine both the binary and continuous variables. Not only did we derive the valid inequalities, we also considered the associated separation problems, that is, given the fractional LP relaxation of the MIP formulation, find the valid inequality that is violated (most) by the current LP solution. For the most important classes of valid inequalities, we derived separation algorithms, some exact, some heuristic.

Next to the valid inequalities, we also derived some preprocessing rules and logical constraints. These rules can be applied in any node of the Branch & Bound tree and exploit the fact that one or more binary variables are set to one.

In order to test the effectiveness of the various classes of inequalities, we did some computational experiments in which we added all the inequalities of a class to the formulation and then solved to model with the commercial solver CPLEX. Moreover, we did experiments with a cutting plane approach in which we separated violated valid inequalities in the root node and then solved the resulting problem with CPLEX. From the preliminary computational results, we may conclude that the

inequalities that only involve binary variables are not that effective. More effective are the inequalities on continuous variables and the inequalities that combine binary and continuous variables. However, as the results of Tables 6.1 to 6.5 show, a cutting plane approach, which adds only violated inequalities, works best. In most cases, solving the instance using this cutting plane approach requires only a fraction of the nodes and computation time, compared to the situation in which we add no valid inequalities.

Chapter 7

Conclusions and further research

In the last chapter of this thesis, we will summarize the results and make some recommendations for further research

7.1 Conclusions

Containers can be considered as the transport revolution of the previous century. Containers turned cargo into a standard commodity. This led to the development of specialized handling equipment which increased productivity at ports enormously. Moreover, increased size of container vessels led to economies of scale. All this, made intercontinental transport of goods cheap, allowing for the globalization of production we know nowadays.

Container terminals form the link between the transport of containers over sea and the hinterland for which the containers are destined. However, also in ports, space, personnel and equipment is expensive. Operations should therefore be carried out as efficiently as possible. Moreover, the high operating costs of seagoing vessels put an increasing pressure on terminals to increase efficiency, in order to preserve the economies of scale of the large container vessels. In this thesis, we gave a detailed overview of the work that has already been done in the field of Operation Research in order to optimize terminal operations.

In Chapters 3 to 6, we considered the scheduling of handling equipment at an automated container terminal. The first of a series of automated container terminals was put into operation by the Rotterdam based company ECT in 1993. Since the

handling equipment is automated, and therefore drivers experience is missing, efficient scheduling is crucial to achieve satisfactory overall performance of the terminal. The layout of the terminal, especially the way in which the AGVs drive, has a large influence on the optimization models that can be applied to schedule the equipment. In this thesis, we considered two types of layouts. The first layout is similar to the current situation at the ECT terminals. In this layout, the AGVs drive in a loop, all passing a common point after they have been unloaded by one of the QCs. However, in the design process of next generation terminals, one also considers layouts in which this is not the case, i.e., the AGVs do not drive in a loop (see e.g. Evers *et al.* (1997)). This usually results in shorter driving distances for the AGVs.

For the case in which the AGVs drive in a loop and pass a common point after unloading by the QCs, we developed a fast Branch & Bound algorithm. This algorithm essentially enumerates all the assignment orders, i.e., the order in which the containers are assigned to the idle AGVs. It is shown that there always exists an optimal solution in which the order of the containers on the individual ASCs is in accordance with the assignment order. Hence, enumerating over all assignment orders gives us the optimal solution. In order to speed up the enumeration, a number of combinatorial lower bounds is calculated in each node of the search tree. Although these bounds are relatively easy to calculate, they are very effective. Computational results show that optimal or near optimal solutions can be found within a reasonable computation time, even for large sized real-life problems.

Based on the Branch & Bound algorithm, we also developed a heuristic Beam Search algorithm. This Beam Search algorithm only exploits a part of the search tree. This is done by following a breadth-first-search strategy and expanding only a limited number of nodes (the beam width) at each level of the tree. The nodes that are further expanded are selected on two criteria, their lower bound on the optimal solution, and in case of ties, their upper bound, which represents a heuristic (feasible) solution. Moreover, we refined the algorithm by introducing a filter, which already disposes the most unpromising nodes at each level in the search tree. As a result, the evaluation of the lower and upper bound has to be done for only a small number of nodes, decreasing the computational effort. The results of the computational tests with the Beam Search algorithm show that this algorithm performs similar to the Branch & Bound algorithm. However, the main advantage of the Beam Search algorithm is that it uses less time.

In Chapter 5, we applied the Beam Search algorithm in a dynamic setting in which we assumed a rolling horizon and stochastic handling times. In this chapter, we also discussed several dispatching rules. These dispatching rules are known from literature, but cannot applied in a straightforward manner. They have to be adjusted

in order to deal with the specific characteristics of an automated container terminal.

More general layouts of the AGV area were considered in Chapter 6. In this chapter, we gave a Mixed Integer Programming (MIP) model for scheduling the handling equipment. Moreover, for the MIP model, we derived several classes of valid inequalities. Connected with the valid inequalities, we also discussed the separation algorithms that can find the valid inequalities that are violated by a fractional LP solution. Computational tests show that some classes are very effective in reducing the computational effort to solve the MIP model. Moreover, using a cutting plane approach in the root node only, gives quite good results. However, the size of the instances that we can solve with this formulation is somewhat limited compared to the instances we can solve with the Branch & Bound algorithm. One may therefore conclude that although a more general layout of the AGV area may result in shorter driving distances, and therefore higher productivity, this productivity gain may be reduced significantly, since the resulting scheduling problem becomes more difficult to solve.

7.2 Directions for further research

The following areas are worthwhile for further investigations.

As already mentioned several times in this thesis, the fact that AGVs cannot load and unload containers themselves makes the schedules of the AGVs and QCs/ASCs highly interdependent. One may argue that for instance an automated lifting vehicle (ALV) may overcome these problems and as a result, the scheduling problems may become easier and not so sensitive to deadlock situations. However, this still does not allow for decomposing the scheduling problem into the scheduling of the ASCs and the scheduling of the ALVs. Although an ALV may be able to load and unload the containers it transports by itself, there is still the situation that there is only limited buffer capacity at the interface with the cranes, especially at the QCs. So, the situation of zero buffers (AGVs) in which the AGVs block the ASCs and the QCs block the AGVs, will result in a situation of limited buffers (ALVs), in which the blocking property is still present! Hence, also in case of ALVs, integrated scheduling of the handling equipment seems the only way to go.

Instead of considering each container as an individual container in the ship (according to the stowage plan), one may group containers of similar size, destination, weight, etcetera. Especially for the future (and even larger) container vessels, this will be relevant. Such a division into categories of containers, allows for more freedom during the loading process of a vessel. For instance, if a container of a certain category is needed at the QC and the category is still available in a number of stack

lanes, one may choose to retrieve the container from the stack lane of which the ASC has the less work to do. However, note that this complicates the decision problem, since in the end, all the containers of a certain category have to be loaded onto the vessel. So, initially there is a lot of choice in which container to choose from which category, but as the loading operation proceeds, the choice becomes more and more limited. Some work on this subject has already been done by Winter (1999), who combines the scheduling of straddle carriers together with the stowage planning of the vessel.

We have not explicitly dealt with reshuffle moves. We assumed that the reshuffles are carried out by the ASC at the time the container is to be retrieved from the stack, i.e., the handling time of a container includes the time required for (possible) reshuffles. The reshuffles can be incorporated in the models by introducing them as containers with zero handling time on the AGV and QC. Moreover, additional precedence constraints have to be taken into account. However, the model which we presented in Chapter 4, does not need to be adjusted structurally. Recall the assignment order π which represents the schedule. Then for a reshuffle i' belonging to container i, we should have that i' precedes i in the assignment order, otherwise the assignment order is not feasible. Since the AGV and QC handling times of the reshuffle are zero, this does not influence the AGVs. Although the incorporation of reshuffle moves is relatively easy, as we illustrated above, it may lead to an increase in computation time, since in principal, the problem size and thus the number of assignment orders we have to enumerate, grows accordingly with the number of reshuffles. Hence, it may be interesting to investigate whether there is a more elegant way of dealing with these reshuffle moves.

One of the ways to increase productivity of QCs that terminal operators are planning to use in the near future, is the so-called twin-lift concept. This means that a QC will load two 20 ft. containers at the same time, i.e., two 20 ft. containers are more or less considered as a single 40 ft. container. However, this means that also the AGV has to deliver both 20 ft. containers at the same time at the QC. This will complicate the scheduling problem of the handling equipment, since an AGV may have to pick up the two containers at different stack lanes.

So far, we have only considered the loading operation of a vessel. However, it is possible to load and unload containers at the same time. For instance, one QC is loading and another QC is unloading. In this case, the model as presented in Chapter 6 can still be applied, although it has to be modified slightly. While loading and unloading at the same time, it may happen that an ASC first handles a container to has be loaded and immediately after this, a container that is unloaded. Hence, for the ASC tasks, we have to introduce switch-over times as well, similar to the

switch-over times we defined for the AGV tasks.

In Chapter 5, we derived two conditions sufficient for a schedule for the AGVs and ASCs to be feasible. Based on these conditions, we developed a number of dispatching rules. Additional research may focus on determining necessary conditions which can then be used to refine the developed dispatching rules and also to speed up the enumeration in for instance, the Branch & Bound algorithm.

Bibliography

Al-Kazily, J. (1983). Productivity at marine-land container terminals. *Transportation Research Record*, 907:57–61.

Ashar, A. (1991). On selectivity and accessibility. *Cargo Systems*, June:44–45.

Avriel, M., Penn, M., and 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., and Witteboon, S. (1998). Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research*, 76:55–71.

Balas, E. and Fischetti, M. (1999). Lifted cycle inequalities for the asymmetric traveling salesman problem. *Mathematics of Operations Research*, 24:273–292.

Balas, E., Fischetti, M., and Pulleyblank, W.R. (1995). The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68:241–265.

Behera, J.M., Diamond, N.T., Bhuta, C.J., and Thorpe, G.R. (2000). The impact of job assignment rules for straddle carriers on the throughput of container terminals. *Journal of Advanced Transportation*, 34:415–444.

Böse, J., Reiners, T., Steenken, D., and Voss, S. (2000). Vehicle dispatching at seaport container terminals using evolutionary algorithms. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, IEEE, Piscataway.

Chadwin, M.L., Pope, J.A., and Talley, W.K. (1990). *Ocean container transportation: an operational perspective*. Taylor & Francis, New York.

Chen, C-Y., Chao, S-L., and Hsieh, T-W. (2000). A time-space network model for the space resource allocation problem in container marine transportation. Dept. of Transportation Management, National Cheng Kung University Tainan, Taiwan.

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

Containerisation International Online (2001). WWW page. Url: http://www.ci-online.co.uk.

Cullinane, K. and Khanna, M. (2000). Economies of scale in large containerships; optimal size and geographical implications. *Journal of Transport Geography*, 8:181–195.

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

De Castilho, B. (1992). Selectivity and accessibility revisited. *Cargo Systems*, June:39–40.

De Castilho, B. and Daganzo, C.F. (1993). Handling strategies for import containers at marine terminals. *Transportation Research B*, 27B:151–166.

Dekker, R., Voogd, P., Nagy, L., and Meersmans, P. (1999). Famas-newcon: Long-term stacking expiriments for the reference case. Technical Report EI-9944/A, Econometric Institute, Erasmus University Rotterdam.

Demir, E.K., Leong, T-Y., Li, C-L., Ng, J.W.C., and Simchi-Levi, D. (1998). Locating containers in a mega terminal. Dept. of IE & MS, Northwestern University/PSA Corporation Limited/ John M. Olin School of Business, Washington University.

Dijkhuizen, G. and Faigle, U. (1993). A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 69:121–130.

Duinkerken, M.B., Evers, J.J.M., and Ottjes, J.A. (2001). A simulation model for integrating quay transport and stacking policies automated container terminals. In *Proceedings of the 15th European Simulation Multiconference (ESM2001)*, SCS, Prague, ISBN 1-56555-225-3.

Duinkerken, M.B. and Ottjes, J.A. (2000). A simulation model for automated container terminals. In *Proceedings of the Business and Industry Simulation Symposium (ASTC 1999)*, ISCS, Washington, ISBN 1-56555-199-0.

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

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

Evers, J.J.M., Van der Wielen, G.J.M., and Duinkerken, M.B. (1997). Terminal layout vanuit de logistiek. Technical Report S-97/2, Trail Research School, Delft. (In Dutch).

Gambardella, L.M., Rizzoli, A., and Zaffalon, M. (1998). Simulation and planning of an intermodal container terminal. *Simulation*, 71(2):107–116.

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

Grötschel, M. and Padberg, M.W. (1977). Lineare Charakterisierungen von Traveling Salesman Problemen. *Zeitschrift für Operations Research*, 21:33–64. (In German).

Imai, A., Nagaiwa, K., and Chan, W.T. (1997). Efficient planning of berth allocation for container terminals in asia. *Journal of Advanced Transportation*, 31:75–94.

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

Kim, K.H. (1997). Evaluation of the number of rehandles in container yards. *Computers and Industrial Engineering*, 32:701–711.

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

— (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. and Kim, H.B. (1999*a*). Segregating space allocation models for container inventories in port container terminals. *International journal of Production Economics*, 59:415–423.

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

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

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

Kim, K.Y. and Kim, K.H. (1997). A routing algorithm for a single transfer crane to load export containers onto a containership. *Computers and Industrial Engineering*, 33:673–676.

— (1998). The optimal determination of the space requirement and the number of transfer cranes for import containers. *Computers and Industrial Engineering*, 35:427–430.

Kozan, E. (1997). Increasing the operational efficiency of container terminals in Australia. *Journal of the Operational Research Society*, 48:151–161.

— (2000). Optimising container transfers at multimodal terminals. *Mathematical and Computer Modeling*, 31:235–243.

Kozan, E. and Preston, P. (1999). Genetic algorithms to schedule container transfers at multimodal terminals. *International Transactions in Operations Research*, 6:311–329.

Kurstjens, S.T.G.L., Dekker, R., Dellaert, N.P., Duinkerken, M.B., Ottjes, J.A., and Evers, J.J.M. (1996). Planning of inter terminal transport at the maasvlakte. In *Proceedings of the 2nd TRAIL congress*, TRAIL Research School, Delft/Rotterdam.

Lai, K.K. and Shih, K. (1992). A study of container berth allocation. *Journal of Advanced Transportation*, 26:45–60.

Legato, P. and Mazza, R.M. (2001). Berth planning and resources optimisation at a container terminal via discrete event simulation. *European Journal of Operational Research*, 133(3):537–547.

Lim, A. (1998). The berth planning problem. *Operations Research Letters*, 22:105–110.

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

Meersmans, P.J.M., Van Hoesel, C.P.M., and Wagelmans, A.P.M. (2001). An integer programming approach to the integrated scheduling of handling equipment at automated container terminals. (Forthcoming).

Meersmans, P.J.M., Vis, I.F.A., De Koster, R., and Dekker, R. (1999). Famas-newcon: een model voor korte termijn stacking. Technical Report EI-9942/A, Econometric Institute, Erasmus University Rotterdam. (In Dutch).

Meersmans, P.J.M. and Wagelmans, A.P.M. (2001*a*). Dynamic scheduling of handling equipment at automated container terminals. Technical Report EI 2001–33, Econometric Institute, Erasmus University Rotterdam.

— (2001*b*). Effective algorithms for integrated scheduling of handling equipment at automated container terminals. Technical Report EI 2001–19, Econometric Institute, Erasmus University Rotterdam.

Murty, K.G., Liu, J., Wan, Y-W., Zhang, C., Tsang, M.C.L., and Linn, R. (2000). DSS (Decision Support Systems) for operations in a container shipping terminal. Working paper, University of Michigan, Ann Arbor.

Nemhauser, G.L. and Wolsey, L.A. (1988). *Integer and combinatorial optimization*. John Wiley & Sons, New York.

Nishimura, E., Imai, A., and Papadimitriou, S. (2001). Berth allocation in the public berth system by genetic algorithms. *European Journal of Operational Research*, 131:282–292.

Ottjes, J.A., Duinkerken, M.B., Evers, J.J.M., and Dekker, R. (1996). Robotised inter terminal transport of containers. a simulation study at the rotterdam port area. In *8th European Simulation Symposium (ESS 1996), Genua 1996*, SCS, ISBN 1-56555-099-4.

Ottjes, J.A. and Veeke, H.P.M. (1999). Simulation of a new port ship interface concept for intermodal transport. In *11th European Simulation Symposium (ESS 1999), Erlangen 1999*, SCS, ISBN 1-56555-177-X.

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

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

Preston, P. and Kozan, E. (2001*a*). An approach to determine storage locations of containers at seaport terminals. *Computers & Operations Research*, 28:983–995.

— (2001*b*). A tabu search technique applied to scheduling container transfers. *Transportation Planning and Technology*, 24:135–153.

Queyranne, M. (1993). Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263–285.

Queyranne, M. and Schulz, A.S. (1994). Polyhedral approaches to machine scheduling. Technical Report 408/1994, Technische Universität Berlin.

— (1995). Scheduling unit jobs with compatible release dates on parallel machines with nonstationary speeds. In Balas, E. and Clausen, J., editors, *Integer Programming and Combinatorial Optimization, proceedings of the fourth IPCO conference, Copenhagen, Denmark, may 1995*, Springer, Berlin.

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

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

Sculli, D. and Hui, C.F. (1988). Three dimensional stacking of containers. *OMEGA*, 16:585–594.

Silberholz, M.B., Golden, B.L., and Baker, E.K. (1991). Using simulation to study the impact of work rules on productivity at marine container terminals. *Computers & Operations Research*, 18(5):443–452.

Sinclair, M. and Van Dyk, E. (1987). Combined routing and scheduling for the transportation of containerized cargo. *Journal of the Operational Research Society*, 38:487–498.

Statistics Singapore (2001). WWW page. Url: http://www.singstat.gov.sg.

Steenken, D. (1992). Fahrwegoptimierung am Containerterminal unter Echtzeitbedingungen. *OR Spektrum*, 14:161–168. (In German).

Steenken, D., Henning, A., Freigang, S., and Voss, S. (1993). Routing of straddle carriers at a container terminal with the special aspect of internal moves. *OR Spektrum*, 15:167–172.

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

Thienel, S. (1995). *ABACUS - A Branch-And-CUt System*. Ph.D. thesis, Universität zu Köln.

Van der Ham, R. (2001). Personal communication.

Van der Meer, J.R. (2000). *Operational control of internal transport*. Ph.D. thesis, Erasmus University Rotterdam.

Van Hee, K.M., Huitink, B., and Leegwater, D.K. (1988). Portplan, decision support system for port terminals. *European Journal of Operational Research*, 34:249–261.

Van Hee, K.M. and Wijbrands, R.J. (1988). Decision support system for container terminal planning. *European Journal of Operational Research*, 34:262–272.

Van Zijderveld, E.J.A. (1995). *A structured terminal design method*. Ph.D. thesis, Delft University of Technology.

Veeke, H.P.M. and Ottjes, J.A. (1999). Detailed simulation of the container flows for the ipsi concept. In *11th European Simulation Symposium (ESS 1999), Erlangen 1999*, SCS, ISBN 1-56555-177-x.

Vickerman, M.J. (1998). Next-generation container vessels. TR News, may - june.

Vis, I.F.A. (2002). *Planning and control concepts for material handling systems*. Ph.D. thesis, Erasmus University Rotterdam.

Vis, I.F.A. and De Koster, R. (1999). Transshipment of containers at a container terminal: an overview. Technical report, Rotterdam School of Management, Erasmus University Rotterdam.

Vis, I.F.A., De Koster, R., Roodbergen, K.J., and Peeters, L.W.P. (2001*a*). Determination of the number of automated guided vehicles required at a semi-automated container terminal. *Journal of the Operational Research Society*, 52:409–417.

Vis, I.F.A., De Koster, R., and Savelsbergh, M.W.P. (2001*b*). Minimum vehicle fleet size at a container terminal. Technical Report ERS-2001-24-LIS, Erasmus Research Institute of Management.

Watanabe, I. (1991). Selection process. *Cargo Systems*, March:35–37.

Wilson, I.D. and Roach, P.A. (1999). Principles of combinatorial optimization applied to container-ship stowage planning. *Journal of Heuristics*, 5:403–418.

Wilson, I.D., Roach, P.A., and Ware, J.A. (2001). Container stowage preplanning: using search to generate solutions, a case study. *Knowledge-Based Systems*, 14:137–145.

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

Wong, P.J., Grant, A.R., and Curley, R.G. (1983). Tandem: Marine and container terminal simulation model. *Transportation Research Record*, 907:27–31.

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.

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

Zaffalon, M., Rizzoli, A.E., Gambardella, L.M., and Mastrolilli, M. (1998). Resource allocation and scheduling of operations in an intermodal terminal. In *10th European Simulation Symposium and Exhibition, Simulation in Industry, Nottingham, United Kingdom, October 26-28*.

Nederlandse samenvatting

De container kan worden beschouwd als *de* transportrevolutie van de afgelopen eeuw. Vrijwel alle producten kunnen in containers worden vervoerd. Naast de standaard containers geschikt voor traditioneel stukgoed, zijn er speciale containers voor koeltransport, bulk en chemicaliën. Door de container is het vervoeren van vracht gestandaardiseerd en dientengevolve zijn de vrachtkosten sterk afgenomen. Met name de Aziatische economieën hebben hiervan geprofiteerd. Immers, door de lage vrachtkosten is het aantrekkelijk geworden om in deze landen te produceren. Het is derhalve geen toeval dat de opmars van de container gelijke tred houdt met de opmars van de "Aziatische Tijgers".

De voordelen van containers ten opzichte van traditioneel stukgoed zijn legio. In containers zijn goederen beter beschermd tegen weersinvloeden en transportschade. Het belangrijkste voordeel is echter dat containers standaard afmetingen hebben. Dientengevolge wordt de overslag enorm vergemakkelijkt en is er een enorme stijging in de productiviteit van overslagterminals. Waar vroeger tientallen havenarbeiders nodig waren om een schip te laden, wordt het werk nu uitgevoerd door enorme kranen die worden bediend door een handjevol mensen. Gevolg: lagere overslagkosten. Hierdoor is sinds zijn uitvinding in de jaren zestig, het aandeel van de container in de totale vervoerstroom sterk toegenomen. De toename van dit volume heeft geleid tot forse schaalvergroting in het zeetransport van containers. Waar in de jaren zestig schepen slechts enkele tientallen TEU (twenty foot equivalent unit; een container van ongeveer 6 meter lengte) konden vervoeren, zijn de huidige zeeschepen in staat om tot 8.000 TEU te vervoeren. Hierdoor zijn er enorme schaalvoordelen te behalen. Echter, de kosten van deze schepen zijn minstens zo omvangrijk. Daarom dienen de schepen zo kort mogelijk in de haven te verblijven om te laden en te lossen. Immers, enkel wanneer het schip vaart kan er geld verdiend worden. Dientengevolge ontstaat er een toenemende druk op terminals om de overslag sneller uit te voeren. Een van de manieren om het laden en lossen van schepen te versnellen is het efficiënt gebruiken van het overslagmaterieel. Hierdoor is het mogelijk om met hetzelfde materieel, in

dezelfde tijd, meer containers over te slaan. Dit draagt bij aan de noodzakelijke productiviteitsverhoging van containerterminals.

Containerterminals vormen de schakel tussen verschillende modaliteiten. Zo wordt door een zeeschip een enorme hoeveelheid containers aan wal gebracht die vervolgens naar het achterland worden afgevoerd met behulp van vrachtwagens, treinen en binnenvaartschepen. Omgekeerd leveren deze laatstgenoemde modaliteiten weer de containers aan die via de zee vertrekken naar een andere regio/continent. Een van de belangrijkste elementen van een containerterminal is de stack. Hierin worden de containers die via de zeezijde binnenkomen tijdelijk opgeslagen in afwachting van hun vertrek naar het achterland. Omgekeerd worden in de stack ook de containers bestemd voor de zeezijde tijdelijk opgeslagen. Naast de stack zijn er enorme kades, uitgerust met kranen, waaraan de zeeschepen geladen en gelost worden. Daarnaast zijn er aparte afhandelingscentra voor vrachtwagens, treinen en binnenvaartschepen. Op een containerterminal is verschillend overslagmaterieel aanwezig. Bijvoorbeeld voertuigen voor het transport van containers tussen de kade en de stack, of stapelkranen voor het neerzetten en oppakken van containers in de stack.

In dit proefschrift worden wiskundige modellen en oplosmethoden beschreven waarmee de efficiëntie op containerterminals kan worden vergroot. Dit is met name van belang op geautomatiseerde terminals. Geautomatiseerde terminals maken gebruik van automatisch geleide voertuigen (AGV's) voor het transport van containers van de kade naar de stack (en vice versa). In de stack worden de containers op elkaar gestapeld met behulp van geautomatiseerde stapelkranen. Aangezien er op de voertuigen en de kranen geen bestuurder aanwezig is, en daardoor het menselijk inzicht ontbreekt, is het van groot belang dat de voertuigen en kranen zo efficiënt mogelijk worden aangestuurd. De aansturing vindt plaats door een planning te maken. Gegeven deze planning krijgt elk voertuig en elke kraan een lijst met opdrachten die in een bepaalde volgorde dienen te worden uitgevoerd. Doel van dit proefschrift is te komen tot een betere planning met behulp van wiskundige technieken.

In hoofdstuk 2 van dit proefschrift wordt uitvoerig ingegaan op de bijdragen die de wetenschap tot nu toe heeft geleverd aan het optimaliseren van containeroverslag met behulp van wiskundige modellen. Er komt een veelheid van onderwerpen aan de orde. Van strategische beslissingen zoals de layout van de terminal en de keuze van het overslagmaterieel, tot planningsproblemen op operationeel niveau. Doel van dit hoofdstuk is om de lezer een inzicht te geven over de stand van zaken en om de huidige literatuur te structureren.

Hoofdstuk 3 beschrijft de centrale doelstelling van dit promotieonderzoek: het optimaliseren van de handelingen op een geautomatiseerde containerterminal door middel van het efficiënter plannen van het overslagmaterieel. De moeilijkheid hierbij

is om de handelingen van verschillende soorten materieel beter op elkaar af te stemmen. Om tot een betere planning te komen, worden in dit proefschrift een aantal wiskundige modellen en bijbehorende oplosmethoden gepresenteerd. Om de resultaten van deze modellen te testen, is een gedetailleerd simulatiemodel ontwikkeld. Het simulatiemodel bootst de werkelijkheid op de terminal na en zodoende kan de effectiviteit van de modellen en methoden getest worden. Dit simulatiemodel wordt kort beschreven in hoofdstuk 3. Tevens wordt in dit hoofdstuk nader ingegaan op het belang van een zogenaamde geïntegreerde planning. Dat wil zeggen dat, in plaats van overslagmaterieel separaat te plannen (bijvoorbeeld het afzonderlijk plannen van automatische voertuigen en het afzonderlijk plannen van automatische kranen), er beter een geïntegreerde planning kan worden gemaakt. Hierdoor worden zogenaamde deadlocks vermeden. Deadlocks doen zich voor wanneer de planning van voertuigen en kranen niet op elkaar is afgestemd. Als gevolg hiervan kan het voorkomen dat de terminal (gedeeltelijk) moet worden stilgelegd om de deadlock op te heffen, met alle nadelige gevolgen voor de productiviteit.

Het geïntegreerd plannen van kranen en voertuigen is rekenkundig zeer complex. Dit houdt in dat wanneer we op zoek gaan naar een optimale planning, in theorie, dit excessief veel rekentijd kan kosten, zelfs op de meest moderne computers. Om de rekentijd in te perken, wordt er in hoofdstuk 4 een Branch & Bound algoritme beschreven. Dit algoritme gebruikt een zoekboom om alle mogelijke oplossingen af te tellen. Op die manier wordt dan de optimale oplossing gevonden. Het Branch & Bound algoritme dat in dit proefschrift wordt gepresenteerd is toegesneden op het probleem van het geïntegreerd plannen van verschillende soorten overslagmaterieel. Hoewel in het slechste geval de rekentijd van dit algoritme nog steeds groot kan zijn, is met behulp van een groot aantal experimenten aangetoond dat in de praktijk het algoritme goed werkt, dat wil zeggen: de rekentijd is beperkt. Een andere manier om de rekentijd te beperken is, om in plaats van naar de optimale oplossing te zoeken, op zoek te gaan naar een goede heuristische oplossing. Dat wil zeggen, een oplossing die niet te ver van het wiskundig optimum verwijderd is. Hiertoe is op basis van het Branch & Bound algoritme een Beam Search heuristiek ontwikkeld. Deze heuristiek beschouwt slechts een beperkt gedeelte van alle mogelijke oplossingen. Rekenkundige experimenten tonen aan dat in de meeste gevallen een zeer goede oplossing wordt gevonden en dit in een beperkte rekentijd.

In de praktijk worden planningsproblemen vaak opgelost met vuistregels. Deze vuistregels zijn eenvoudig en daardoor makkelijk te implementeren. Echter, wanneer ze worden toegepast voor het plannen van overslagmaterieel op geautomatiseerde containerterminals, garanderen ze geen deadlock vrije planning. In hoofdstuk 5 worden een aantal van deze vuistregels beschreven en er worden voorwaarden afgeleid

waaraan de vuistregels moeten voldoen opdat ze onder alle omstandigheden een correcte planning geven. De prestatie van de vuistregels wordt vergeleken met de prestatie van de Beam Search heuristiek uit hoofdstuk 4. Het blijkt dat de prestatie van de Beam Search heuristiek in de meeste gevallen veel beter is dan de prestatie van de (aangepaste) vuistregels. Slechts voor enkele van deze vuistregels zijn de verschillen beperkt.

Hoofdstuk 6 beschouwt een algemeen wiskundig model voor het geïntegreerd plannen van voertuigen en kranen op een geautomatiseerde containerterminal. Dit model kan worden opgelost met behulp van commerciele software. Echter, door de rekenkundige complexiteit van het probleem kunnen slechts problemen van beperkte grootte worden opgelost. Om toch grotere problemen te kunnen oplossen, kan het model worden uitgebreid met zogenaamde geldige ongelijkheden. Deze ongelijkheden beperken de zoekruimte en dientengenvolge wordt de rekentijd beperkt. Voor de gegeven wiskundige formulering wordt een groot aantal klassen van deze geldige ongelijkheden afgeleid. De effectiviteit van de diverse klassen van ongelijkheden is getest. Uit de rekenresultaten blijkt dat het toepassen van geldige ongelijkheden een aanzienlijke reductie in rekentijd kan opleveren.

Tot slot wordt in hoofdstuk 7 het proefschrift kort samengevat. Tevens worden er in dit hoofdstuk nog enkele ideeën voor vervolgonderzoek gegeven.

The Tinbergen Institute is the Institute for Economic Research, which was founded in 1987 by the Faculties of Economics and Econometrics of the Erasmus Universiteit Rotterdam, Universiteit van Amsterdam and Vrije Universiteit Amsterdam. The Institute is named after the late Professor Jan Tinbergen, Dutch Nobel Prize laureate in economics in 1969. The Tinbergen Institute is located in Amsterdam and Rotterdam. The following books recently appeared in the Tinbergen Institute Research Series:

223. B. VAN DER KLAAUW, *Unemployment duration determinants and policy evaluation.*

224. F. MEDDA, *The assembled city: Analyses of multiple urban dimensions.*

225. A.F. TIEMAN, *Evolutionary game theory and equilibrium selection.*

226. R.R.P. KOUWENBERG, *Dynamic asset liability management.*

227. J.S. SIDHU, *Organization mission, business domain orientation and performance: A conceptual and empirical inquiry.*

228. G. ROMIJN, *Economic dynamics of Dutch construction.*

229. M.C. VERSANTVOORT, *Analysing labour supply in a life style perspective.*

230. J.J.J. GROEN, *Testing multi-country exchange rate models.*

231. C.F.A. VAN WESENBEECK, *How to deal with imperfect competition: introducing game-theoretical concepts in general equilibrium model of international trade.*

232. M.L. NDOEN, *Migrants and entrepreneurial activities in peripheral Indonesia. A socioeconomic model of profit-seeking behaviour.*

233. L.A. GROGAN, *Labour market transitions of individuals in eastern and western Europe.*

234. E.G. VAN DE MORTEL, *An institutional approach to transition processes.*

235. P.H. VAN OIJEN, *Essays on corporate governance.*

236. H.M.M. VAN GOOR, *Banken en industriefinanciering in de 19e eeuw. De relatie tussen Mees en Stork, Van den Bergh gaat naar Engeland.*

237. F.R.M. PORTRAIT, *Long-term care services for the Dutch elderly. An investigation into the process of utilization.*

238. M. VAN DE VELDEN, *Topics in correspondence analysis.*

239. G. DRAISMA, *Parametric and semi-parametric methods in extreme value theory.*

240. I.F.C. MULDER, *Soil degradation in Benin: Farmers' perceptions and responses.*

241. A.W. SALY, *Corporate entrepreneurship. Antecedents and consequences of entrepreneurship in large established firms.*

242. S. VAN VELZEN, *Supplements to the economics of household behavior.*

243. R.A. VAN DER GOOT, *High performance linda using a class library.*

244. E. KUIPER, *The most valuable of all Capital. A gender reading of economic texts.*

245. P. KLIJNSMIT, *Voluntary corporate governance disclosures; An empirical investigation of UK practices.*

246. P.J.G. TANG, *Essays on economic growth and imperfect markets.*

247. H. HOOGEVEEN, *Risk and insurance in rural Zimbabwe.*

248. A.J. VAN DER VLIST, *Residential mobility and commuting.*

249. G.E. BIJWAARD, *Rank estimation of duration models.*

250. A.B. BERKELAAR, *Strategic asset allocation and asset pricing.*

251. B.J. VAN PRAAG, *Earnings management; Empirical evidence on value relevance and income smoothing.*

252. E. PEEK, *Discretion in financial reporting and properties of analysts' earnings forecasts.*

253. N. JONKER, *Job performance and career prospects of auditors.*

254. M.J.G. BUN, *Accurate statistical analysis in dynamic panel data models.*

255. P.C. VERHOEF, *Analyzing customer relationships: Linking attitudes and marketing instruments to customer behavior.*

256. C.S. BOS, *Time varying parameter models for inflation and exchange rates.*

257. A. HEYMA, *Dynamic models of labour force retirement; An empirical analysis of early exit in the Netherlands.*

258. S. DEZELAN, *The impact of institutional investors on equity markets and their liquidity.*

259. D.J. DEKKER, *Network perspectives on tasks in account management.*

260. F.G. VAN OORT, *Agglomeration, economic growth and innovation. Spatial analyses of knowledge externalities in the Netherlands.*

261. U. KOCK, *Social benefits and the flow approach to the labor market.*

262. D.J. BEZEMER, *Structural change in Central European agriculture. Studies from the Czech and Slovak Republics.*

263. D.P.J. BOTMAN, *Globalization, heterogeneity, and imperfect information.*

264. H.C. VAN DER BLONK, *Changing the order, ordering the change. The evolution of MIS at the Dutch railways.*

265. K. GERXHANI, *The informal sector in transition. Tax evasion in an institutional vacuum.*

266. R.A.J. BOSMAN, *Emotions and economic behavior. An experimental investigation.*

267. A. P. VAN VUUREN, *Empirical analysis of job search using novel types of data.*

268. H. VAN DE VELDEN, *An experimental approach to expectation formulation in dynamic economic systems.*

269. L. MOERS, *Institution, economic performance and transition.*

270. N.K. BOOTS, *Rare event simulation in models with heavy–tailed random variables.*