Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Decomposition methods and rolling horizon approach for the yard crane scheduling problem

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Sander van Dijk

Delft, The Netherlands
May 2015

MSc THESIS APPLIED MATHEMATICS

"Decomposition methods and rolling horizon approach for the yard crane scheduling problem"

SANDER VAN DIJK

Delft University of Technology

Daily supervisor

Dr.ir. M. Burger

Responsible professor

Prof.dr.ir. K.I. Aardal

Other thesis committee members

Dr.ir. R.J. Fokkink

May 2015

Delft, The Netherlands

Abstract

Yard crane scheduling is one of the operational optimization problems that arise in a port container terminal. Multiple cranes have to store containers in the yard or retrieve containers from the yard. The scheduling of these cranes is complex and it is difficult to find an optimal schedule within limited computation time.

In this thesis multiple mixed integer programming formulations are introduced that can be used to model the yard crane scheduling problem. The first formulation fixes zones for a certain planning window. Yard cranes are assigned an individual zone and these zones are non-overlapping. The second model uses a time discretization. We assume that yard cranes can handle 1 job in each time interval of 3 minutes. Working zones for the yard cranes can overlap, but not in the same or subsequent time intervals.

Decomposition methods are used to reduce the problem complexity and reduce computation times. A logic-based Benders decomposition is used for the first model were the assignment of cranes to jobs is done separately from finding an optimal path for the yard cranes individually. Iterating between those problems can reduce computation times when the number of iterations is small. After decomposition, the size of these partial problems is much smaller than the size of the problem before decomposition. Some additional lower bound procedures are introduced to further speed up computation. A lower bound procedure can identify and exclude an assignment that will lead to a bad schedule in an early stage. When the problem instances are large enough, logic-based Benders decomposition showed a reduction in computation times.

The problem is modeled in a rolling horizon fashion and time decomposition is introduced to again reduce computation times. Instead of trying to solve the problem for one long time period, this time period is split into two or more small time intervals. These intervals are solved separately where connection between them is solved by iterating between those individual time intervals. The rolling horizon reflects the practice in container terminals better since business goes on the entire day and operational decisions are made continuously, over a limited time horizon. By scheduling for a certain time period ahead, schedules can be made online without using all data available in the future. This keeps the computation times limited since scheduling a whole day at once would be too computationally expensive.

The arrival times of jobs at the yard are not deterministic. Exact arrival times of trucks at the yard are hard to predict. Therefore some stochastic models are desirable. In this thesis we introduce uncertain arrival times of trucks at the gate of the terminal. Exact arrival times are only known a few minutes ahead while other arrival times follow an exponential distribution. We solve this model using expected values of the arrival times as well as by simulating possible scenarios.

Acknowledgements

This thesis describes my research done at TBA regarding yard crane scheduling. This research is the final step to obtain my master's degree in Applied Mathematics at Delft University of Technology. I came in contact with TBA at a business diner organised by my rowing club D.S.R. Proteus-Eretes. I have spent many hours for training and other activities at my rowing club during my entire period as a student, which makes it a very important part of my student life. At this business diner, Martijn Coeveld of TBA introduced me to container terminals and their operational challenges. I would like to thank Martijn for getting me interested in yard crane scheduling and offering this research opportunity to me.

In addition, I would like to thank my supervisors Mernout Burger and Karen Aardal for providing guidance during the entire thesis project. Their feedback helped me to realize in what direction my research should continue and also their comments on how to write a thesis have been very useful for me. Writing has never been one of my stronger points, so their feedback on structure, spelling and grammar was very valuable. Furthermore I would like to thank Robbert Fokkink, for taking seat in my thesis committee and Liselot Arkesteijn for reading many parts of my thesis and giving extensive feedback on spelling and grammar.

Finally, I would like to thank some groups of people. First of all, my colleagues from TBA for sharing their knowledge on yard crane scheduling and container terminals. And secondly, a number of friends for showing interest in my research. This helped me structure my conclusions and prepare my final presentation.

Contents

Chapter 1

Introduction

Nowadays transport of cargo over sea is mostly done via container transport. Containers are transported on large vessels overseas and these vessels are loaded and unloaded at container terminals. At the quayside of a terminal, large quay cranes move the containers from vessel to quay or vice versa. In the storage yard, large yard cranes stack containers upon each other. These containers are stored until they have to be transported further. The transportation between storage yard and quay cranes is done via internal trucks. These trucks are either driven by employees of the terminal, or they are automated guided vehicles. Transportation between the storage yard and the gate on the landside is done via external trucks and/or internal trucks. External trucks are trucks that are driven by chauffeurs who are not employed at the terminal. They take containers for further transport over roads. Internal trucks can be used for containers that need to be transported to a railway or to inland vessels.

Many operational problems arise in the container terminals. Good schedules are needed for the quay cranes, the yard cranes and the internal trucks. Many obstacles arise in avoiding collision and minimizing delay or downtime of these equipments. Most operational aspects are complex by themselves, such as the scheduling of yard cranes and quay cranes. Those operational problems (among others) are known to be NP-hard. The notion of NP-hard will be explained in Chapter 3 but suggests that it is not possible to find an algorithm that solves this scheduling problem efficiently. As a result, the size of the problem (number of jobs and yard cranes) is very important for the solution time. Combining all operations in a container terminal to a total model including container flow, truck dispatching and crane scheduling is very complex.

This research was conducted at TBA, a consultancy company founded in the Netherlands in 1996. TBA specializes in terminal operations and develops software for automated guided vehicles in terminals. Part of their business is simulating container terminal performance under different circumstances, to consult on daily operations or long term decisions. Long term decisions are , for example, the choice of equipment in the terminal, size of the terminal, layout of the terminal or other strategic decisions. They use opti-

mization approaches for different operations. This is done to advise container terminal operators on how to improve their performance. Some of these approaches contain an advanced yard crane scheduler which can be implemented in the existing terminal software or their emulation models. Their emulation models are (part of) virtual terminals that simulates real time container terminal operations.

1.1 Problem definition

This thesis will focus on the scheduling of yard cranes. At the container yard, containers are moved from trucks to the stack and vice versa. These actions are defined as jobs or moves. Sometimes reshuffling is needed before the right container can be reached. Jobs arrive at different times, at different places and in some cases with different expected handling time. There can be different type of jobs with different priority. The input for the problem consists of the number of cranes, number of bays and a list of jobs containing: ready time, handling time, bay number (location) and type of job (see Figure 1.1 (left)). Each job needs to be assigned to a yard crane. Therefore a schedule for the yard cranes needs to be made up such that every job is handled by a yard crane. With this input a schedule has to be made for the yard cranes. Each yard crane is assigned a subset of the jobs such that the sum of delays for the jobs is minimized (see Figure 1.1 (right)). The delay of a job is defined as the difference between the completion time of a job in the found schedule and its originally planned completion time. Its originally planned completion time is the time the job becomes available (because a vehicle arrived at the yard) plus its handling time (processing time). This objective (minimizing delay) results in a minimal waiting time for vehicles at the yard. Other possible objectives for the *yard crane scheduling problem* are minimizing the traveling distance of yard cranes or minimizing the makespan. Since yard cranes share the same lane along the stack, they cannot cross each other. An important constraint for scheduling the yard cranes is this non-crossing constraint. Furthermore, jobs cannot be handled before a truck arrives at the right location (the job ready time). This problem is an NP-hard problem (see Chapter 4) and therefore not efficiently solvable unless $P = NP$. To deal with this NP-hardness issue, input cannot be too large. The theoretical worst-case runtime of exact algorithms will grow exponentially with the problem size. In practice, it is sometimes possible to solve for medium sized problems in a reasonable amount of time.

Because of many interacting activities in container terminals, the expected arrival time of jobs is rather uncertain except for the very near future. Truck arrival times are only assumed to be reasonably stable for the upcoming hour. This makes it impossible to plan all operations far ahead, so schedules need to be made online for the near future. As a consequence, algorithms used for the scheduling need to be fast and find good schedules in a limited amount of time.

This problem shows similarities to the well known combinatorial optimization problems: *vehicle routing problem*, *multiple traveling salesman problem* and *assignment problem*.

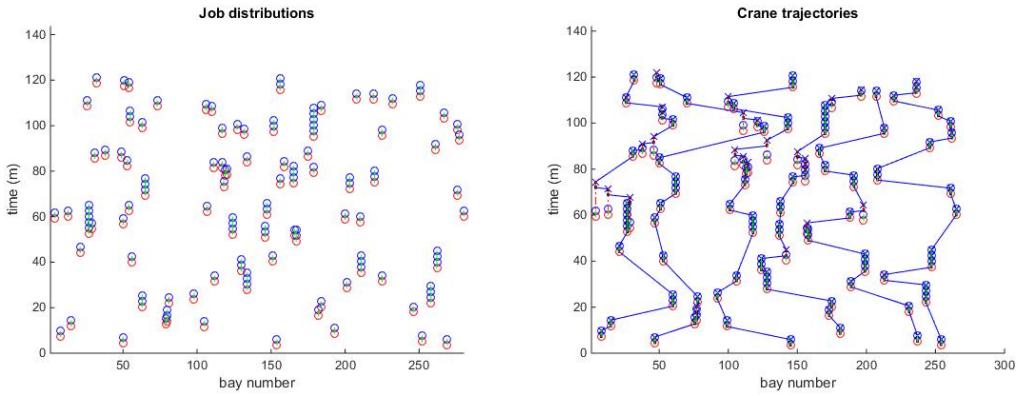


Figure 1.1: Input (left), Output schedule (right)

A route for the yard cranes must be found where all jobs (cities) are performed (visited) once. There are time constraints for the jobs on the route and additional spatial constraints concerned with the non-crossing of the cranes. This last part is the main difference between the yard crane scheduling problem and the vehicle routing problem or multiple traveling salesman problem. In view of the assignment problem, the jobs need to be assigned to the different yard cranes. Again, some extra spatial constraints are added for the non-crossing of the yard cranes.

1.2 Research goals

The main purpose of this research is to identify important modeling factors and solution algorithms to construct a good yard crane scheduler. A good yard crane scheduler can construct schedules fast such that with a limited number of yard cranes the waiting time of vehicles at the yard is minimized. A minimal vehicle waiting time of trucks at the yard is necessary for a good quay crane performance rate, which is the most important performance measure for container terminals.

We will use mixed integer linear programming (MILP) formulations for modeling the problem. The class of MILP problems belongs to the class of NP-hard problems. This means that it is impossible (assuming P$\neq$NP) that an algorithm can be found that solves an arbitrary MILP formulation in polynomial time. However, this does not mean that all MILP problems are difficult to solve. Since the *yard crane scheduling problem* is NP-hard, the corresponding MILP formulations are unlikely to be solved efficiently. For very large problems it is therefore ambitious to solve the problem to optimality. On the other hand, NP-hardness is a classification for the complete class of *yard crane scheduling problems*. This does not mean that every problem instance is difficult to solve. Small problem instances can be solved in a limited amount of time. Using some special structure, sometimes larger problem instances are also solved easily. Recent research in this area

is mainly focused on finding heuristic solutions. We want to investigate whether it is possible to come up with good MILP formulations that can be solved to optimality in a reasonable amount of time. Specific problem characteristics and good MILP formulations might make solving in reasonable time possible in practice. Reasonable time would be two minutes to solve problem instances with around 30 jobs.

First we assume that the truck arrival times at the yard are deterministic. However, as a result of traffic jams or vehicle failure, the truck arrival times are uncertain. Stochastic programming can be used to model this uncertainty. We will investigate, to what extent we can incorporate this within the yard crane scheduler.

1.3 Solution methods

To solve the MILP problems we formulate in this thesis, we make use of the software packages Gurobi and Matlab. Matlab is used as an interface for creating the problem instances (matrices) and returning and evaluating the results. Matlab is a programming language with interactive environment that can be used for a wide variety of mathematical activities. We use Matlab for defining the MILP problems, call on Gurobi for solving, iterating between different MILP problems, read and write data files and analyzing results. Gurobi is used as solver for the MILP formulations. It is a commercial software package originally designed for solving LP problems. Gurobi solves MILP problems using branch and bound and LP relaxations.

1.4 Research contribution

The main research contribution of this thesis is the introduction of MILP formulations for the yard crane scheduling problem. We introduce two new MILP formulations and adjust one other MILP formulation. We tested the different formulations for their applicability. These formulations are compared on computation times and their found objective values. Furthermore, Benders decomposition is introduced for the different MILP formulations. Logic-based Benders decomposition will further reduce computation times for the first model.

A rolling horizon approach is introduced to deal with long planning periods (more than 15 minutes) or online planning problems. The rolling horizon approach is a way to solve only a part of the problem, without creating “bad” boundaries between the different parts. At a specific point in time the problem is solved for a short time window (for example the next 15 minutes). After a short time (for example 5 minutes) the problem is solved again. We also introduce a time decomposition that can be used to solve long planning periods. Instead of solving the whole planning period at once, the planning period is divided into two or more pieces that are solved separately. The separate problems are solved iteratively to optimize over global delay instead of delay per time

interval. Different parameters and different MILP formulations are tested within this rolling horizon approach.

At the end of this thesis also stochastic input is introduced. Two algorithms are introduced to deal with uncertain arrival times. For these algorithms the aforementioned time decomposition is used. The performance of these algorithms is tested using exponential distributions for the arrival times.

1.5 Outline

The first four chapters give an introduction to the yard crane scheduling problem. In Chapter 2, some introduction to container terminals is given. The different parts of a container terminal are described as well as the operational problems that arise. In Chapter 3 an introduction to some mathematical concepts is given. The concept "optimization problem" and the concept of an algorithm are described. Furthermore, we define an MILP problem and describe the "branch and bound" solution method. In addition, we give a framework for a probability space which is used in Chapter 8 for stochastic input. Chapter 4 contains the literature review. Here we summarize briefly the research that is done in container terminals and in more detail for the *yard crane scheduling problem*. Some models that use similar approaches as the approach in this thesis are described in more detail.

Chapters 5 to 8 describe several formulations and solution methods for the yard crane scheduling problem. In Chapter 5 we introduce the models we use for the *yard crane scheduling problem*. Three different models (MILP formulations) are given and compared on their number of variables and constraints. In Chapter 6 we introduce Benders decomposition. Benders decomposition is used to alter the MILP formulations to reduce the computation time for solving the problem. Because new containers (and therefore storage or retrieval jobs at the yard) keep on arriving during the day, we need a rolling horizon approach to solve this scheduling problem. This rolling horizon approach is given in Chapter 7. In Chapter 8 we shift from deterministic to stochastic input. Some initial ideas on how stochastic input can be integrated in the deterministic models are presented. It can be used as a starting point of a more advanced study to integrate stochastic input into the models.

Results on delay and computation times for the different models are presented in Chapter 9. These results are presented both for the static case as well as the rolling horizon. In the last section of Chapter 9, also some results on stochastic input are presented. Chapter 10 gives the conclusion of this thesis. In Chapter 11 some validation as well as further recommendations are presented.

Chapter 2

Container terminal

In this chapter the lay-out of a typical container terminal is described in more detail. In Section 2.1 we describe the different parts of the container terminal. After that, Section 2.2 describe the operational problems concerned with the different parts of the container terminal. In Section 2.3 we briefly discuss different kind of port container terminals and their influence on operational decisions. This chapter is focused on the container terminal as a whole. Operations in the yard are very dependent on the rest of the terminal operations. A more detailed study of the *yard crane scheduling problem* has been incorporated into Chapter 4.

2.1 Layout of a container terminal

Transport of commodities over sea is mostly done in cargo containers. In a container terminal, the containers are transshipped between different transport vehicles, such as ships, trains and trucks. Most containers have standardized sizes and are 20 or 40 feet long. The throughput of containers in a terminal, also known as vessel capacity, is measured in TEU (Twenty-feet Equivalent Unit). A container of 40-feet or larger is accounted for as 2 TEU.

Container terminals contain one or multiple berths (places where vessels can dock). These berths are located lengthwise a quay. At the quay, huge quay cranes are positioned to load or unload the vessels (see Figure 2.1). These quay cranes can be moved along the quay to change position, but they are fixed on a rail so they cannot cross each other. Multiple quay cranes can be assigned to one vessel in order to speed up the unloading and loading. A vessel consists of multiple holds lengthwise and only one quay crane can work at a hold.

Sometimes unloading and loading are separate activities and the loading starts only

Figure 2.1: Quay cranes (from TBA)

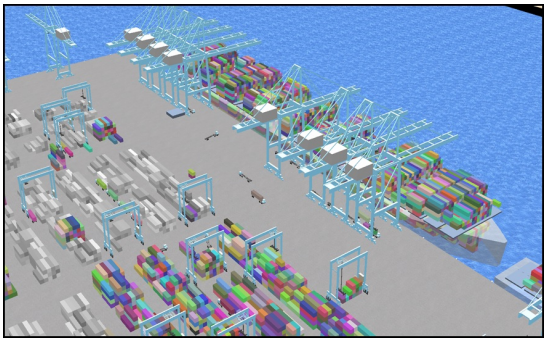


Figure 2.2: Terminal layout (from TBA)

when unloading is finished. The operation can be accelerated by using dual cycling. In this case loading activities are embedded in the unloading activities to reduce empty horizontal movements of the quay crane. A container is picked up from the ship and moved to an internal truck at the quay. Another container from another internal truck is picked up and moved to the ship. In this way the spreader of the quay crane does not need to move empty from quay to ship after every move.

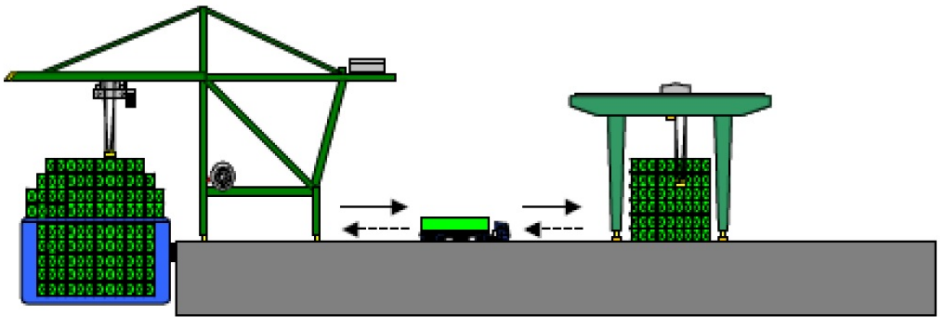


Figure 2.3: Terminal layout schematic (Ng and Mak (2005))

When unloading a ship, the quay crane transfers the containers to internal trucks. These trucks drive the containers to the storage yard where the containers are stored for later activities by yard cranes (see Figure 2.2 and Figure 2.3). In some terminals automated guided vehicles (AGVs) are used for internal transport. From the storage yard containers are transferred to the gate for further transport over land by external trucks, trains or inland shipping. For loading a ship the flow of events is reversed. This process is illustrated in Figure 2.4.

Some ports are used as transshipment ports. In transshipment ports containers are unloaded from a vessel, stored in the yard and later loaded onto another vessel. The ratio between export and import varies for different ports. Import or export influences the type of operations in the terminal.

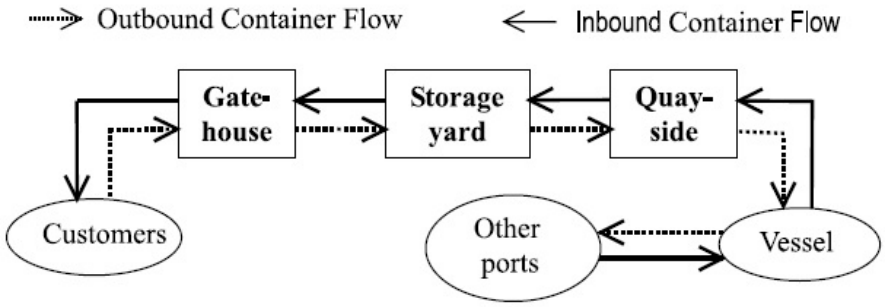


Figure 2.4: Flow of containers through the terminal (Zhang et al. (2002))

Yard cranes are used to store or retrieve containers from the storage yard. There are two main types of yard cranes, the RTG (Rubber Tyred Gantry) crane and the RMG (Rail Mounted Gantry) crane. Note that this research is focused on the RTG crane. The problem for the RMG crane is closely related but different on some essential parts. The models developped later are only applicable for RTG cranes. A similar approach might work for the RMG crane but this is not included in this research. The RTG crane has the ability to turn 90 degrees and is not restricted by rail. It can gantry (move in its entirety along a lane to another bay), but can also switch lanes by turning 90 degrees twice. An RMG crane cannot leave a lane since it is fixed on straight rails. For both cranes a different yard layout is used. For an RTG crane, container lanes are orientated parallel to the quay (see Figure 2.2). A container is transferred from a pile to the side of the bay (see Figure 2.5 perpendicular to the quay. Therefore the RTG crane does not have to gantry during a job. A lane typically consists of a number of blocks of about 45 bays. The space between blocks is used by trucks to get to the gate or quay crane and by the RTG crane to turn to switch to a block in another lane (see Figure 2.6. For RMG cranes the yard layout is different. The lanes are perpendicular to the quay and each block has at both ends a spot where containers are brought or picked up by yard trucks. So when the RMG crane picks up a container from a slot, it has to gantry to the end of the block to deliver the container to the truck. This is why the scheduling has to be done differently. Typically two RMG cranes work together in a block, both serving one end of the block. Systems other than RMG cranes or RTG cranes for retrieving containers from the yard are known, but these have less similarities with our RTG scheduling problem.

2.2 Operational problems in the container terminal

Operations in a container terminal can be modeled using a mathematical problem description. This problem can be split into different subproblems (see Figure 2.7). Usually these problems are solved separately and then combined in a service plan for the complete terminal. Integrated approaches are desirable, but difficult to solve. At the quay

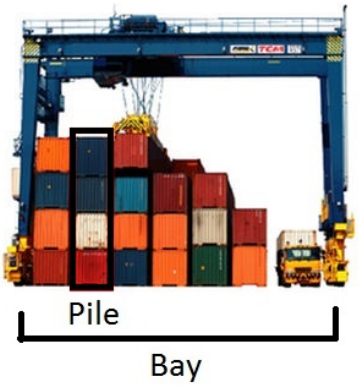


Figure 2.5: RTG crane

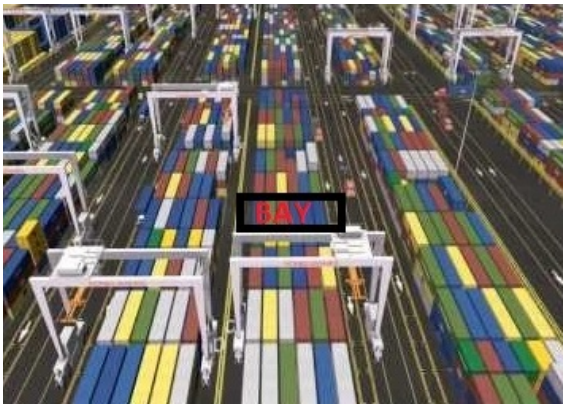


Figure 2.6: RTG terminal block

two problems can be identified which are sometimes combined. First of all there is the *berth allocation problem (BAP)*, in which it has to be decided which ship moors at which berth. Next, the quay cranes have to be distributed over the quay and assigned loading and unloading jobs for the ships. This is called the *quay crane scheduling problem (QCSP)*. From the quay crane, containers have to be transported to the yard. This is called the *vehicle routing problem (VRP)* or *internal truck scheduling problem (ITSP)*. It should not be confused with the combinatorial optimization problem that is also called the vehicle routing problem. Both problems share some properties, but are not similar.

In general, a group of internal trucks is assigned to a specific quay crane. This is called pooling. Those internal trucks have to bring containers to the yard or pick up containers from the yard for that quay crane. Next to the assignment of containers to trucks, also the route of the internal trucks has to be decided to avoid possible traffic crowding.

To know the destination of the internal trucks in the yard, a storage plan is needed. This is sometimes called the *storage allocation problem (SAP)* or *stacking problem*. It is used to determine which container has to come at which position (block, row, bay, pile, tire). A good storage plan has to minimize the reshuffles needed when retrieving a container from the yard, but should also spread workload through the yard to prevent very high workload in one block.

When a storage allocation plan and the internal trucks are assigned, the *yard crane scheduling problem (YCSP)* is to be solved. The yard cranes have to perform several jobs in the yard during a time period. A job can be retrieval, storage or reshuffle and has a position, a ready time and an expected handle time. Yard cranes have to divide work such that the waiting time of the internal trucks is minimized. They are restricted from crossing each other within a lane and the number of yard cranes available is limited.

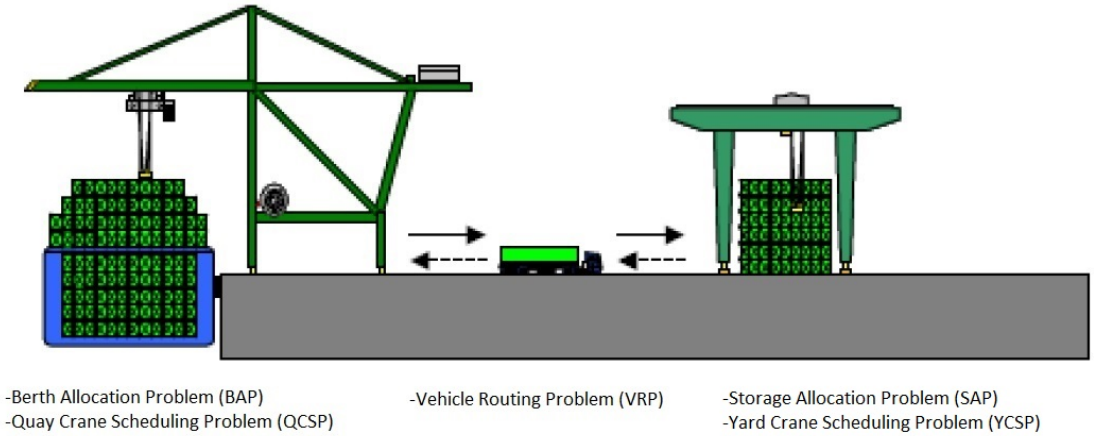


Figure 2.7: Operational problems in the container terminal

2.2.1 Objectives

The main objective of a container terminal is the turn around time of vessels. This is widely accepted as the main performance measure for container terminals. The turn around time for a vessel is the time a vessel spends in the terminal. Unloading and loading of the vessel needs to be as fast as possible. Mathematically this means that the makespan of loading and unloading activities for a vessel is minimized. The makespan is the time needed to complete a group of activities or jobs. For the *berth allocation problem* this means that the average turn around time per vessel is minimized. Each vessel is assigned to a berth that will result in a short makespan for that vessel, but should consider other vessels. Allocating a small ship to a berth with much equipment (quay cranes) might result in a short makespan for that vessel, but results in a very long makespan for a larger vessel that needs to dock at a berth with fewer equipment.

Once a ship is allocated to a berth, the objective for the *quay crane scheduling problem* is to minimize the makespan. This is the part where loading and unloading activities are completed as soon as possible. For the activities on the landside of the quay, this means that the quay crane is supported as good as possible. Once the quay cranes are assigned to the containers on the vessel, it is important that the containers are picked up at the quay crane on time (or arrive at the quay crane on time in case of loading). This means that the delay per jobs is minimized instead of the makespan (total completion time). For the *vehicle routing problem* this means that there should be enough vehicles available and the probability that a vehicle arrives to late at the quay crane is minimized. For the *yard crane scheduling problem* this results in minimizing the delay for the vehicles. A vehicle arriving at the yard needs to be served with minimal delay.

Although the above are the main objectives, sometimes other objectives are important

too. For example, if slightly reducing the turn around time of a vessel would result in the use of much more expensive equipment, then this is not desirable. Reducing the travel distances of equipment or other cost reducing operations can be a side objective.

2.3 Container terminal type

There are three different kind of container terminals with respect to their function. Some terminals are mainly focused on export. This is for example the case for many Asian container terminals. Commodities arrive from the landside of the terminal and leave on the seaside. On the contrary, there are terminals that are mainly focused on import. Containers arrive via the seaside and are further transported into the hinterland. The third category for a container terminal is transshipment terminals. These terminals are used to redistribute containers from one ship to another. Containers are unloaded from a vessel to the yard of the container terminal. After a short period (five days on average) the containers are loaded onto another vessel. Transshipment terminals are becoming more common nowadays. These different kind of container terminals result in different kind of operational issues and objectives. For export container terminals, loading activities have a high priority. This results in different strategies for container stacking and yard crane dispatching. Containers intended for the same vessel are stored close together such that they can be loaded onto the vessel easily. For transshipment terminals the focus lies on storage allocation. Because vessels need to be loaded and unloaded at the same time, it is difficult to store containers at a convenient bay or block. This might lead to a lot of undesired reshuffling. Minimizing the reshuffling and dwell time (time that a container spends in the yard) is very important for transshipment terminals.

Chapter 3

Optimization problems

In this chapter some mathematical terminology and concepts are introduced. A few important concepts will be discussed in detail, while for other concepts a high-level introduction is given. Furthermore, references to the relevant literature are provided.

3.1 Optimization problem

An optimization problem is a problem of finding a best solution within a certain solution space. There is a set of variables which can attain different values. We call the variables of an optimization problem decision variables since their value in general correspond to a decision in practice. The decision variables of an optimization problem can be brought together in a vector. Let $\mathbf{x}$ denote the vector of decision variables. The best values for the decision variables are determined by minimizing (or maximizing) an objective function. This corresponds to choosing the best decision. The combined values that the decision variables can attain is limited by a set of constraints. So not all values of the variables are feasible. Optimization is about finding the best values for the decision variables that are feasible under these constraints.

Minimize Objective
subject to Constraints

Let $\bar{\mathbf{x}}$ be a realization of $\mathbf{x}$. If $\bar{\mathbf{x}}$ does not conflict any of the constraints, then $\bar{\mathbf{x}}$ is called a feasible solution. The feasible set of an optimization problem is the set $F = \{\mathbf{x} : \mathbf{x} \text{ is a feasible solution}\}$. The optimal solution (usually denoted by $\mathbf{x}^*$) is the feasible solution that results in the smallest value (outcome of objective function) in case of minimization. For maximization the feasible solution resulting in the largest objective value is referred to as the optimal solution. There is a wide variety of optimization problems, but in this thesis we will consider linear and mixed integer linear programming problems.

3.1.1 Linear programming

In linear programming (LP) the objective function is a linear function and also the constraints are linear inequalities (or equalities). Assume $\mathbf{x}$ is an n-dimensional vector of decision variables. We assume all vectors are defined as column vectors unless stated otherwise. Let $\mathbf{c}$ be an n-dimensional vector of constants, $\mathbf{A}$ be an m by n-dimensional matrix and $\mathbf{b}$ be an m-dimensional vector. We define the following linear programming problem:

$$\begin{array}{lrcl} \text{minimize} & \mathbf{c}^T\mathbf{x} & & \\ \text{subject to} & \mathbf{A}\mathbf{x} & = & \mathbf{b} \end{array}$$

In this LP problem, $\mathbf{A}$ is called the constraints matrix. Together with the vector $\mathbf{b}$ it sets constraints on the decision variables $\mathbf{x}$. The vector $\mathbf{c}$ is called the objective vector, with $\mathbf{c}^T$ we denote the transpose of $\mathbf{c}$. Instead of the equality constraints we could also have inequality constraints. We could also have additional non-negativity constraints on the $\mathbf{x}$-variables. The problem would for example look like this:

$$\begin{array}{lrcl} \text{minimize} & \mathbf{c}^T\mathbf{x} & & \\ \text{subject to} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

Many other variations are possible. If the reader is interested in more possible variations or a standard form, he/she is referred to Bertsimas and Tsitsiklis (1997).

LP problems are usually solved via the simplex method. This method is described in Bertsimas and Tsitsiklis (1997) Chapter 3. With the simplex method, an optimal solution can be found fast in many cases. In this thesis we focus on the formulation of the linear programming problems. Solving the formulations is done via advanced software packages. A clear understanding of the simplex method or other solving methods is therefore not necessary for the understanding of this thesis.

3.1.2 Mixed integer linear programming

For linear programming problems only continuous decision variables are used. A possible extension is to also include binary or integer variables. Integer variables take only integer values while binary variables only attain the value 0 or 1. When including integer variables, we lose the property of LP that makes solutions easy to find. The restriction to integer values makes the problems much more difficult to solve. On the other hand, the restriction to integer or binary variables gives a lot of modeling possibilities.

Many decisions in operational problems are binary. Deciding to do something or not is easily modeled by a binary variable, e.g. the binary variable takes the value 1 when the activity is done and the value 0 when it is not. Often binary and continuous decision variables are combined. Let $\mathbf{x}$ again denote a vector of continuous decision variables

and let $\mathbf{y}$ denote a vector of binary decision variables. The following is an example of a mixed integer linear programming (MILP) problem.

$$\begin{array}{lll} \text{minimize} & \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} & \\ \text{subject to} & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} & \leq \quad \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} & \\ & \mathbf{y} \in \{0, 1\} & \end{array}$$

In this example, $\mathbf{A}$ and $\mathbf{B}$ are constraints matrices, the vectors $\mathbf{c}$ and $\mathbf{d}$ are objective vectors and the vector $\mathbf{b}$ a constraint vector. MILP problems are usually solved using a branch and bound method. This is a search tree method. A short description of this method is given in Section 3.3. In Chapter 5 we will formulate the *yard crane scheduling problem* as an MILP problem.

3.1.3 LP-relaxation

To solve an MILP problem often an LP-relaxation is used. In an LP-relaxation, the binary or integer restrictions on the decision variables are discarded. The solution space of the MILP problem is a subset of the solution space of the LP-relaxation. This means that every solution to an MILP problem is also a solution for its LP-relaxation. The converse is not true in general. Let Z^*_{LP} denote the optimal value of the LP-relaxation and Z^*_{MILP} the optimal value of the MILP problem. We must thus have $Z^*_{LP} \leq Z^*_{MILP}$. If the input for an optimization problem is fixed, we speak of a problem instance. In the case of linear programming, this means that the constraint matrix $\mathbf{A}$, the right-hand side vector $\mathbf{b}$ and the objective vector $\mathbf{c}$ are specified. The integrality gap is defined as follows (I denotes a problem instance):

$$\text{Integrality gap} := \sup_I \frac{Z^*_{MILP}(I)}{Z^*_{LP}(I)}$$

This is an important measure for the quality of an LP-relaxation. If the integrality gap is small, we speak of a strong LP-relaxation. When solving an MILP, often lower bounds using such an LP-relaxation are used (see Section 3.3). If the integrality gap is small, the lower bound becomes more useful. We define the relative integrality gap for a specific problem instance I as: $\frac{Z^*_{MILP}(I) - Z^*_{LP}(I)}{Z^*_{MILP}(I)}$.

3.2 Complexity

Complexity analysis is an important part of optimization problems. It tells us something about the difficulty to solve a class of optimization problems. Complexity is a large topic in computer science, so we will describe only the most important elements used in this thesis. For formal definitions the reader is referred to Savage (1998) or another source for complexity analysis.

3.2.1 Decision problems

For complexity classes we need the notion of a decision problem. A decision problem has a simple "yes" or "no" solution. Every optimization problem has a corresponding decision problem. Instead of minimizing an objective we ask if there exists a feasible solution such that the objective is smaller than a constant K. This constant K is part of the input for a problem instance. The question whether a feasible solution with objective smaller than K exists can be answered by "yes" or "no".

3.2.2 Algorithms

Algorithms are used to solve optimization problems. In the case of LP problems this is commonly done using the simplex method. In the case of MILP problems this is done via branch and bound. Advanced computer software packages are available that can be used to solve these problems. For linear programming (and mixed integer linear programming) Gurobi and CPLEX are the most used solvers. In this thesis we use Gurobi (and its built-in algorithms) to solve our MILP problems. An algorithm systematically searches for the optimal value of the optimization problem. It is a set of instructions that needs to be followed and executed.

3.2.3 Runtime

An important concept for analysing decision problems (or optimization problems) is their worst-case runtime. The worst-case runtime to solve a decision problem is dependent on the algorithm that is used. It is a performance measure for the algorithm that is used to solve a decision problem and not directly a complexity measure for the decision problem.

This worst-case runtime bound for an algorithm is a function of the input size. Assume the input size of an optimization problem is measured by n. If there exists an n_0 such that for every $n > n_0$ the runtime will not increase more with respect to n than a function $c \cdot f(n)$ (where c is a constant), we say that the runtime is bounded by $f(n)$. For this, the "Big O" notation is used. For example, $\mathcal{O}(n^2)$ means that the runtime will not increase more than quadratically with respect to the input size (provided that the input size is large enough). If the worst-case runtime is bounded by a polynomial function we say that the runtime is polynomially bounded. If the runtime is not bounded by a polynomial function it is usually bounded by an exponential function. Exponential functions increase much faster than polynomial functions when n becomes large (see Figure 3.1).

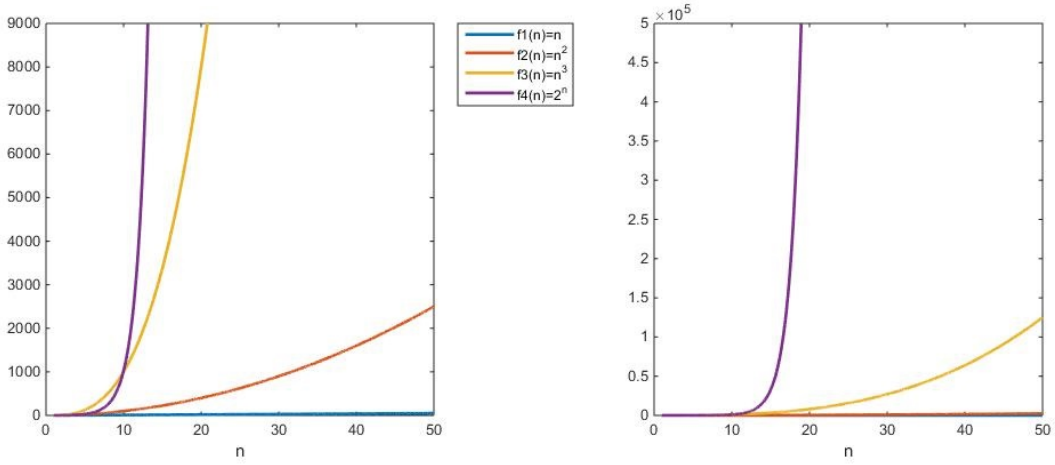


Figure 3.1: Worst-case runtime growth

3.2.4 Complexity classes

For the complexity measurement of the decision problem we use the best known worst-case runtime for an algorithm that can be used to solve this class of decision problems. If two algorithms are known to solve a decision problem and one runs in polynomial time while the other runs in exponential time, we refer to the complexity of the decision problem as polynomial.

Note that complexity of decision problems is based on a class of decision problems and not on one specific instance of this class. A problem instance is a specific realization of this class. If a class of decision problems is hard to solve, this does not mean that every instance of this class is hard to solve. A specific instance which is very small or has some nice characteristics can very well be very easy to solve (short runtime for the algorithm). When referring to an optimization problem (or decision problem) we actually mean the whole class of problem instances.

P and NP

Two important complexity classes are P and NP. A decision problem belongs to the class NP if and only if a yes-instance is verifiable within polynomial time. So if someone presents a realization of the decision variables that leads to a yes outcome, there exists an algorithm that checks within polynomial time that this realization is indeed feasible and leads to an objective smaller than K. In the case of MILP, we only need to check whether the solution fulfills all the constraints and leads to an objective smaller than K. The complexity class P contains all decision problems that can be solved by an algorithm that runs in polynomial time. It is a subclass of the class NP. If a decision problem belongs to the class NP and we have no proof that it also belongs to P, it is

expected that the runtime of an algorithm will be long for problem instances with a large input size.

NP-complete

The class NP-complete is a subclass of the class NP. It is the class of decision problems which are at least as hard as the other decision problems in the class NP. For this class we need the notion of a reduction between decision problems. We say that decision problem A reduces to B when there is a polynomial time reduction from A to B. A polynomial time reduction from A to B means that any problem instance of decision problem A can be rewritten to a problem instance of optimization problem B by an algorithm in polynomial time. This must be an "if and only if" connection between a "yes"-instance for problem A and a "yes"-instance for problem B. So an algorithm for solving decision problem B can be used as a subroutine for solving decision problem A.

A decision problem B from NP belongs to NP-complete, when there exists a reduction from all decision problems in NP to B. Therefore, the subclass NP-complete is referred to as the hardest problems of the class NP. One of the most important open problems in mathematics is the question whether $P = NP$. That would mean that all decision problems that are verifiable in polynomial time, are solvable within polynomial time. If an algorithm for decision problem B from NP-complete would be polynomially bounded in runtime, this would mean that all decision problems from NP could be solved in polynomial time by reduction to B. From this it would follow that $P = NP$. Since no-one has found an algorithm that is bounded in polynomial time for any decision problem in NP-complete, it is widely believed that $P \neq NP$ and hence that no such algorithm exists. An example of a decision problem that belongs to NP-complete is the MILP problem. The *yard crane scheduling problem* can be modeled as a subclass of the MILP problem with certain characteristics as we will see in Chapter 5. That the MILP problem is NP-complete does not mean that every subclass with certain characteristics is NP-complete. There can be subclasses of the MILP problem that are very easy to solve (meaning they can be solved by an algorithm that runs in polynomial time). An example of such an MILP problem is the *assignment problem* (Bertsimas and Tsitsiklis (1997), Sec. 7.8). This is however not the case for the *yard crane scheduling problem* as we will see in Chapter 4. The *yard crane scheduling problem* itself belongs also to the class of NP-complete problems.

NP-hard

Another complexity class that is interesting for this thesis is the class NP-hard. It has similarities with the class NP-complete. A problem C belongs to NP-hard, when there exists a reduction from all decision problems in NP to C. The difference with NP-complete is that C does not have to be in NP itself. So NP-complete is a subclass

of NP-hard (see Figure 3.2). MILP problems as defined above are part of the NP-hard class. Verifying whether a given realization of the variables is the optimal solution is in general not possible in polynomial time. Therefore, an MILP problem is in general not NP in contrast to its corresponding decision version. Verifying whether a given realization of the variables is feasible and has an objective value smaller than K is possible in polynomial time.

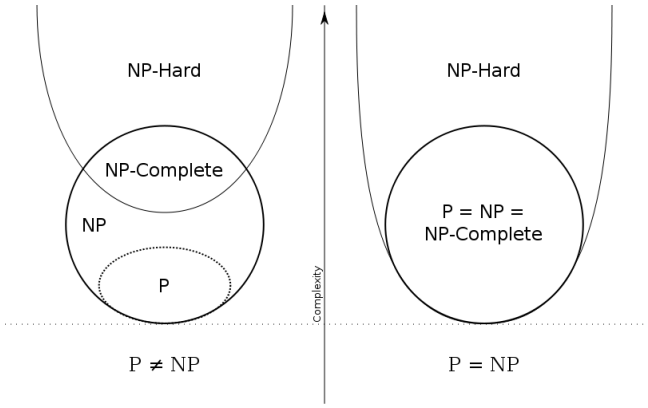


Figure 3.2: Set representation of complexity classes

3.2.5 Approximation

NP-hard optimization problems are often hard to solve in limited computation time. Large problem instances usually take a long computation time to solve. It would not be surprising if computation times, just like the worst-case runtime, would increase exponentially with the input size. For those large problem instances it is sometimes preferable to approximate the solution to the optimization problem. When an algorithm is designed to search for a solution within a certain bound of the optimal solution, we call it an approximation algorithm. If no specific bound for approximation of the algorithm exists, we call the algorithm a heuristic. It finds a feasible solution, but it usually does not find the optimal solution (although this is possible). A heuristic solution can be close to the optimal solution but also very far from it. Approximation algorithms are bounded in polynomial time to approximate the solution of a optimization problem where no polynomial time algorithm is known. If such a polynomially bounded (with respect to the input size) approximation algorithm is guaranteed to find a feasible solution within a factor 2 of the optimal solution, we speak of a 2-approximation algorithm. More general, if the approximation algorithm is guaranteed to find a solution within a factor $1+\delta$ of the optimal solution we speak of a $(1+\delta)$-approximation algorithm. For some optimization problems it is possible to construct for any $\varepsilon > 0$ an algorithm (dependent on ε) that will find a solution within a factor $1+\varepsilon$. If the runtime of such an algorithm is bounded polynomially for the input size we speak of a PTAS (polynomial-time approximation scheme). The runtime does not have to be polynomial in ε, it might be that the runtime

is bounded by a function $n^{\frac{1}{\varepsilon}}$. If the runtime of such an approximation algorithm is even polynomially bounded in $\frac{1}{\varepsilon}$ we speak of an FPTAS (fully polynomial-time approximation scheme). There exists an FPTAS for the *quay crane scheduling problem* with a fixed number of quay cranes (see Section 4.1.1).

The above classification is theoretical and based on worst-case scenarios. The approximation bound is a worst-case bound for example. This means that the approximation algorithm can as well find the optimal solution for some (or even a lot) of problem instances. On the other hand, a polynomial bound does not directly mean it is solvable fast. If ε is very small, in a PTAS the runtime may still be very long. Polynomial boundedness does not guarantee that algorithms have short computational time.

3.3 Branch and bound

For solving MILP problems to optimality, usually a branch and bound method is used. Gurobi also uses a branch and bound method, and we will describe this method briefly in this section (see also www.gurobi.com /resources/getting-started/mip-basics for their own explanation).

3.3.1 Tree structure

Branch and bound uses a so-called tree structure to find an optimal solution (see Figure 3.3). A tree has a root node that is connected to several other nodes. These nodes can again be split into more nodes and so on. The node that is split is referred to as a parent node. The parent node is split to one or several child nodes. If we use branch and bound for an MILP problem, we usually branch a parent node in two different child nodes. When nodes are no longer split we arrive at the bottom of the tree. Those nodes are called leaves. In Figure 3.3 this tree structure is shown.

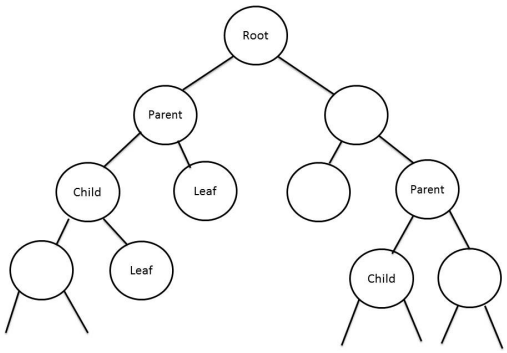


Figure 3.3: Tree structure

3.3.2 Branch and bound algorithms

We explain how a branch and bound algorithm for an MILP problem with binary variables works. A branch and bound algorithm starts in the root node of a tree. The root node contains the set of all feasible solutions to the MILP. We assume for explanatory reasons that we have a minimization problem, but the method is similar for maximization problems. The algorithm will start by using a heuristic to find a good initial feasible solution that works as an upper bound (Z_{up}). This upper bound will be updated during the execution of the remainder of the algorithm. If no feasible solution is found by the heuristic or such an heuristic is unknown, then the upper bound is set to infinity. Defining the first initial upper bound is an initialization step for the algorithm.

Next the LP-relaxation of the MILP problem is solved. If the solution of the LP-relaxation is integral, we can stop because this will also be the optimal solution for the MILP. Otherwise a binary variable is selected on which we will branch. In this case, the root node is split into two child nodes. For the first child node, the binary variable is set to 0 and for the second child node the binary variable is set to 1. The two child nodes contain a new MILP problem that is slightly smaller (less binary variables) than the MILP problem of the root node. Thereby, the union of the solution sets of the child nodes is equal to the solution set of the root node. Next, one of the two child nodes is selected for further evaluation.

For every node, a branch and bound method uses the the so-called bounding and branching procedures. When evaluating a node, the first step is the bounding procedure. Every node consists of an MILP problem (for the root node the original MILP problem). First the LP-relaxation of this MILP problem is solved. Based on the result of the LP-relaxation, either the tree is pruned in this node or it is branched into two new child nodes again. Pruning the tree happens when branching is not necessary anymore. There are three different cases when pruning is possible:

- The LP-relaxation is infeasible
- All variables are integral for the LP-relaxation
- The LP-relaxation provides a lower bound that is larger than the best found upper bound for the original MILP problem.

If the LP-relaxation is infeasible, then also the MILP problem is infeasible. The algorithm can stop evaluating this node. Let Z_{LP} denote the optimal value of the LP-relaxation of the current node when the LP-relaxation is feasible. If the solution to the LP-relaxation is integral, it is also the optimal solution for the MILP. If $Z_{LP} < Z_{up}$, we can update the upper bound with $Z_{up} := Z_{LP}$. Note that the value Z_{up} is global while Z_{LP} holds only for the current node. If the solution to the LP-relaxation is not integral, we have either $Z_{LP} \geq Z_{up}$ or $Z_{LP} < Z_{up}$. If $Z_{LP} \geq Z_{up}$, we can also prune the tree in this node. The value Z_{LP} functions as a lower bound. We have $Z_{MILP} \geq Z_{LP} \geq Z_{up}$ and therefore this node will not lead to a better solution.

If pruning is not possible, we again choose a binary variable for branching. The node is branched into two child nodes, each containing less binary variables. This branching is similar to the branching in the root node. Since the number of variables in an MILP problem is finite, the branch and bound algorithm will prune the tree at some point and we arrive at a leaf node. If all nodes are evaluated, the best found upper bound is our optimal solution.

Because solving LP-relaxations is a fundamental part of this algorithm, LP-relaxations must be solved very fast. This is usually done by the simplex method. This method is not guaranteed to run in polynomial time, but experience learns that it is almost always able to find solutions fast. The worst-case runtime of the simplex method is almost never attained. The branch and bound algorithm works well when branches can be pruned in an early stage such that only a limited amount of nodes requires examination. Therefore it is important that the integrality gap of the LP-relaxation is small. If the integrality gap is large, it is less likely that $Z_{LP} \geq Z_{up}$.

This branch and bound method is used in many algorithms. Different configurations of the method will lead to different algorithms. For instance, we need to specify in which sequence nodes are investigated within the branch and bound method. Furthermore, we need to specify on which binary variable we will branch. An algorithm can search the tree depth-first, breadth-first or something in between. It is important to find a good upper bound quickly, such that nodes can be pruned in an early stage. Important subroutines can identify on which variable the branching should be done and which child node should be examined first. This can all strongly influence the number of nodes that needs to be examined by the algorithm.

3.4 Stochastic programming

In Chapter 8 we will introduce stochastic input for the *yard crane scheduling problem*. To introduce this stochastic input, we need the notion of a probability space and distribution functions. We will not present a precise definition/derivation of all elements, but introduce the various concepts such that the concepts can be understood. We will first introduce a probability space and a probability measure. The derivation is based on Shreve (2004) and is analogous to the derivation in the TU Delft course Financial Mathematics. For the sake of clarity and length some parts are excluded. The reader is referred to Shreve (2004) or another source on probability spaces.

3.4.1 Construction of a probability space

We will construct a probability space on the extended real line $\mathbb{R} \cup \{-\infty, +\infty\}$. We consider $\mathbb{R} = (-\infty, +\infty)$ and the intervals $(a, b] = \{x \in \mathbb{R} : a < x \leq b\}$, with $-\infty \leq a < b < +\infty$, and $(c, +\infty)$, with $-\infty \leq c < +\infty$. Let $\mathcal{A}$ be the class of all subsets in $\mathbb{R}$

that are finite unions of disjoint intervals in $\mathbb{R}$. Let $\mathcal{A}$ also include the empty set $\emptyset$. The class $\mathcal{A}$ is an algebra. A subset $\mathcal{C} \subseteq 2^X$ (where 2^X is the powerset of X) is an algebra if it is non-empty, closed under complementation and closed under finite unions. If the notion of finite unions is extended to countable unions, we speak of a σ-algebra. The smallest σ-algebra that contains $\mathcal{A}$ is the Borel σ-algebra of $\mathbb{R}$. We will denote this Borel σ-algebra by $\mathcal{B}(\mathbb{R})$. $\mathcal{B}(\mathbb{R})$ is the smallest σ-algebra that contains all open intervals. Note that any type of interval is included in this Borel σ-algebra since $(a, b) = \cup_{n=1}^{\infty}(a, b - \frac{1}{n}]$ and countable unions are included in the σ-algebra. We call $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ an equipped space.

Probability measure

To define a probability measure, we first define a repartition function. A repartition function is a function $F : \mathbb{R} \to [0, 1]$ such that:

1. F is non-decreasing, i.e. $F(x_1) \leq F(x_2)$ for every $x_1 < x_2 \in \mathbb{R}$
2. $\lim_{x\to-\infty} F(x) = 0$ and $\lim_{x\to+\infty} F(x) = 1$
3. F is right-continuous, i.e. $\forall x \in \mathbb{R}$, $\lim_{t\to x^+} F(t) = F(x)$.

We define the following probabilities where F is a repartition function:

$$P_0((a, b]) = F(b) - F(a) \quad \text{and} \quad P_0((c, +\infty)) = 1 - F(c)$$

Using the additivity of $\mathcal{A}$ we extend P_0 on the algebra $\mathcal{A}$:

$$P_1(A) := \sum_{i=1}^{n} P_0((a_i, b_i])$$

P_1 is an extension of P_0, we have for $(a, b] = A \in \mathcal{A}$: $P_1(A) = P_0(A)$. The domain of P_0 is the class of intervals while the domain of P_1 is $\mathcal{A}$. P_1 is well-defined in the sense that for different interval representations of a set A, P_1 assigns the same probability. P_1 is a probability on the algebra $\mathcal{A}$. Remember that for P_1 to be a probability, we need:

1. $P_1(\mathbb{R}) = 1$
2. $\forall A \in \mathcal{A}, P_1(A) \geq 0$
3. For $A_1, A_2, \ldots, A_n \in \mathcal{A}$ pairwise disjoint, $P_1(\cup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P_1(A_i)$

The first two follow directly from the first two definitions of repartition function F. The third is a bit more cumbersome to show and is excluded in this thesis. We even need that P_1 is σ-additive on $\mathcal{A}$. This means that point 3 is extended to countable unions. This follows from continuity from above of P_1 but is again further excluded. For the last extension we need the Carathéodory Extension Theorem. This theorem states: Any σ-additive measure defined on an algebra $\mathcal{E}$ can be uniquely extended to the σ-algebra $\sigma(\mathcal{E})$ generated by itself.

For our probability P_1, this means that it can be uniquely extended to the Borel σ-algebra $\mathcal{B}(\mathbb{R})$. This means that there is a unique probability measure P on the equipped space $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ such that $P((a,b]) = P_1((a,b]) = P_0((a,b]) = F(b) - F(a)$. This measure P is called the Lebesque-Stieltjes probability measure. Our equipped space $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ in combination with the probability measure P forms a probability space $(\mathbb{R}, \mathcal{B}(\mathbb{R}), P)$. This described method to define a probability space can be extended to multiple dimension or even countably many via the Kolmogorov extension theorem.

3.4.2 Random variable

We call a random variable any measurable function $X : (\Omega, \mathcal{F}) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$. So a random variable is a function defined on the outcome set Ω (equipped with σ-algebra $\mathcal{F}$), that takes values in $\mathbb{R}$, such that $\{\omega : X(\omega) \in B\} \in \mathcal{F}$, $\forall B \in \mathcal{B}(\mathbb{R})$. A random variable is a function from the abstract outcome space Ω to the much easier to understand observation space $\mathbb{R}$ where we have defined a probability.

3.4.3 Distribution functions

The notion of a random variable allows us to define probability density functions and distribution functions. A probability density function of a random variable X on $\mathcal{B}(\mathbb{R})$ is defined as:

$$P_X(A) := P(\{\omega : X(\omega) \in A\}) = P(X^{-1}A), A \in \mathcal{B}(\mathbb{R}).$$

The distribution function of a random variable X is given by:

$$F_X(x) := P_X((\infty, x]) = P(X^{-1}(-\infty, x]), x \in \mathbb{R}.$$

This probability density function and distribution function allow us to describe real valued variables that reflect some abstract events in the outcome space Ω via the random variable.

Chapter 4

Literature review

The literature review in this chapter is focused on container terminal operations. The first section discusses solutions to the different operational problems in the container terminal. It provides a brief overview of what has been studied in container terminal operations. It also gives some insight in the historical development of research in the container terminal. Although different operational problems arise in the container terminal, similar approaches have been proposed. Many problems have been modeled using MILP and heuristics are proposed to overcome the NP-hardness. The second section focuses is on the *yard crane scheduling problem*. In the context of yard crane scheduling, different types of problem definitions can be identified. We divide these problem definitions into 3 different groups and describe them in a different subsection.
- The dispatching of yard cranes when a certain number of containers from a specific group have to be picked up.
- The deployment of RTG cranes to different blocks in a yard (Inter-block crane deployment problem).
- The dispatching of yard cranes to several jobs within a lane, including interference constraints. The actual framework differs among researchers and is described separately in each subsection.

Some examples of heuristic methods are mentioned in this literature review. Methods as beam search algorithms, genetic algorithms and simulated annealing are mentioned for the reader that is interested in/ familiar with these approaches. We will not explain these methods in this thesis since they take no further part in the remainder of this thesis. It is however important to note the difference between a heuristic approach and a optimal approach (see Subsection 3.2.5).

4.1 Terminal operations

In the past decades a lot of research was conducted on operational activities in container terminals. As the container terminal business has become more competitive, there is a need to optimize operations. There are some comprehensive literature reviews on operations in container terminals, given by Vis and De Koster (2003); Steenken et al. (2004); Stahlbock and Voß (2008); Vacca et al. (2007); Meersmans and Dekker (2001). Some research was conducted to describe an integrated approach of a whole container terminal, combining the quay cranes, internal trucks and yard cranes. Chen et al. (2007) give a mixed integer linear program (MILP) which deploys different machines to different operation phases of jobs. Many researchers advocate the use of an integrated model, but they also refer to the complexity of this approach as a downside. The research that considers the integrated scheduling of machine and material handling systems for a flexible manufacturing system shows similarities to the integrated container terminal optimization problem. They differ on the aspect of the freedom/availability of the machinery after performing a job. For example, the quay crane has to wait for an internal truck before finishing the job and proceed to the next. Bierwirth and Meisel (2010) give some examples of studies where the BAP and the QCSP are combined. More research has been done to berth allocation, quay crane scheduling, internal truck scheduling, yard crane scheduling and stacking policies separately. Below we describe shortly the *quay crane scheduling problem* (which is studied most extensively) and the other operation problems.

4.1.1 Quay crane scheduling problem

Daganzo (1989) describe one of the first researches on operations inside a container terminal. He studies quay crane deployment to different holds of a ship. Since then, much work concerning quay cranes has been performed (for an overview, see Steenken et al. (2004)). The *quay crane scheduling problem* has some similarities to the *yard crane scheduling problem*. Both problems have multiple cranes sharing one lane and deal with a non-crossing constraint. The main objective of a container terminal is the turnaround time of a vessel. This results in different objectives for the quay crane and the yard crane. While the quay cranes have to minimize the makespan (latest completion time) of all the jobs/cranes (Kim and Park (2004)), in general the yard cranes have to minimize the vehicle waiting time in order to let the yard trucks arrive at the quay crane on time. Another difference is the ready times of jobs. For a quay crane certain containers have to preceed other containers (Kim and Park (2004)) or the problem is modeled for a whole hold (Lee et al. (2008)). In both cases, there is no job ready time, every job can be processed directly from start (if not violating constraints). For yard cranes a job cannot be handled before the yard truck arrives. Therefore a job has a job specific ready time given in the input. The *quay crane scheduling problem with non-interference constraints* is shown to be NP-complete (Lim et al. (2004); Lee et al. (2008)). However, Lim et al.

(2004) show that a fully polynomial time approximation scheme (FPTAS) is possible when the number of quay cranes is fixed. The derivation of this algorithm is based on the FPTAS algorithms in Sahni (1976). In their model (Lim et al. (2007)), they do not keep track of the exact position and time of jobs. They process each job with a given process time and gantry time is ignored. To deal with the non-crossing constraint, they let all the cranes handle their jobs left to right after making sure no conflicting jobs are scheduled simultaneously.

4.1.2 Other operational problems

The BAP (Lim (1998)) has also been shown to be NP-complete. For the BAP, the minimum sum of waiting time and handling time for each ship is optimized (Imai et al. (2001)). The vehicle routing problem is addressed by Kim and Bae (2004), they use a network based MILP to determine a one-to-one assignment and in addition a heuristic is proposed. The objective is minimizing the quay crane idle time. Commonly, internal trucks are pooled and a group of three to seven internal trucks are assigned to a quay crane. Cao et al. (2010) study the integrated problem of yard crane scheduling and internal truck scheduling using an MILP. They reduce the LP solution space using Benders decomposition. Benders decomposition uses row generation to progress to a solution. They do not address ready times for jobs and do not take the order in which the containers arrive at the quay crane in account. This *internal truck scheduling problem* is highly connected with the *storage allocation problem* (Lee et al. (2009)). In the *storage allocation problem* the expected number of reshuffles is minimized by assigning storage locations to containers (Hwan Kim and Bae Kim (1999)).

4.2 Yard crane scheduling

The *yard crane scheduling problem* with respect to RTG cranes consist of multiple approaches. Some researches use a grouped container approach for the *yard crane scheduling problem*. In the grouped containers approach, containers are divided into different groups (based on weight and destination). A certain number of containers from a specific group need to be loaded onto a vessel at a certain time interval. This is described in Subsection 4.2.1. The other approach is a one to one approach where every specific container needs to be picked up at a specific moment. Every container is assigned to a specific RTG crane. This is described in Subsection 4.2.3. The problem of deploying different yard cranes to different blocks or lanes is described in Subsection 4.2.2. This inter-block crane deployment is often combined with a crane to container assignment within each block on a different hierarchical level (see for example the model of Guo and Huang, 4.2.3).

4.2.1 Grouped containers

Kim and Kim (1999, 2003); Narasimhan and Palekar (2002); Lee et al. (2007) use container groups to model the yard crane schedule. They introduce several container groups which are stored in one or more bays. The job for the yard crane is to pick up a certain number of containers from every group. The optimization problem is to find a route for the yard crane passing several bays to pick up enough containers. In Kim and Kim (1999) they derive an MILP for the problem which is solved using a dynamic programming algorithm. In Kim and Kim (2003) they give heuristic approaches for their problem, resulting in much faster algorithms. For the interested reader, they use a beam search algorithm and a genetic algorithm where their beam search algorithm outperformed their genetic algorithm for large instances (30 bays). This (grouped container) problem is sometimes referred to as the transtainer (yard crane) routing problem. Jung and Kim (2006) extend the model to serve multiple quay cranes at the same time. Narasimhan and Palekar (2002) prove that this transtainer routing problem is NP-Complete. Furthermore, they prove that there cannot be a polynomial time approximation algorithm with a worst-case lower bound less than $(k^2 + 2k - 1)/2(2k - 1)$ where k is the number of container groups (unless $P = NP$).
Lee et al. (2007) derive a model for two transtainers working in different blocks on the same load plan. They give a heuristic algorithm in the class of SA (simulated annealing) to find good feasible solutions to the problem. It performs around 10% above their lower bound for an optimal solution.

4.2.2 Inter-block crane deployment problem

Linn et al. (2003) were the first to give a model for RTG crane deployment to different blocks. The work load per block is given and it should be decided how many RTG cranes have to be deployed to every block. Only two RTG cranes can work in a block so as to avoid collision. RTG cranes can also move only once per shift. They identify source blocks and sink blocks. Naturally, RTG crane might switch from a source block to a sink block to avoid idle time. An MILP is used to solve this problem. The objective is minimizing workload overflow to the next period. They identify reasonable improvements in yard operations, but suggest that true benefit may be derived when the deployment model is well integrated with the other parts of yard operation control such as grounding strategy and yard traffic control.
Linn and Zhang (2003) extend this model dividing their 4 hour planning horizon in smaller deployment periods. This increases the computation time, so they propose a least cost heuristic. Two conditions are identified which have to be fulfilled to possibly move an RMG crane to another block. Solutions of 3-6% above optimal are found while testing of the algorithm. Results where found within 1 second so they claim that their heuristic is very robust and efficient in producing near optimal solutions.
Murty et al. (2005) add arrival times to this model. They do not assume that all workload is ready at the beginning of a time period.

In Zhang et al. (2002) the MILP (with multiple time-zones) is solved via Lagrangian relaxation. They decompose their MILP into an LP and an MILP similar to a network flow problem. Both problems are quit easy to solve. Additional constraints are added to the system to reduce a large duality gap.
Cheung et al. (2002) gives a Lagrangian decomposition to the problem comparable to the procedure of Zhang et al. (2002). They also rewrite the problem to a non-linear problem with a piecewise linear objective functions. They give a successive piecewise-linear approximation method which slightly outperforms their Lagrangian decomposition. Much faster and better approximations are found than their benchmark for large instances. For their benchmark they try to find an optimal solution using CPLEX, but stop after running CPLEX for one hour and return the best feasible solution found. Cheung et al. (2002) also prove that this problem(inter-block crane deployment problem) is NP-hard in the strong sense.
Recently Sharif et al. (2012) gave an agent-based solution to this NP-hard problem. They match cranes to blocks using different preferences based on distance of the blocks and available workload.

4.2.3 Crane to job assignment

In this problem definition a schedule is to be made, including each container specifically (in contrast to the grouped containers). Instead of picking up containers of a certain type, each container must be scheduled individually. The exact problem definition differs sligthly among researchers. The main simularities are the non-crossing constraints and the specific visiting of a certain bay to perform a job. Three models stick out and are presented in more detail. After that some recent research on genetic algorithms with respect to this problem definition is mentioned shortly.

Model of Ng (and Mak)

Ng and Mak (2005) describe a model for picking up containers in a yard block where only one yard crane is working. Jobs have a ready time and handling time given by the operating system. Every job has a location which is used to identify the travel time between two different jobs. They give an MILP to find the optimal sequence of jobs. As an objective they use the waiting time of trucks or more precisely: $t_i - r_i - h_i$. Here, t_i is the completion time while r_i and h_i are the ready time and handling time respectively. Since this *machine scheduling problem* is NP-complete (Lenstra et al. (1977)), they propose a branch and bound method to solve the problem to optimality. As expected, the computational time increases exponentially with the input size (number of jobs). This branch and bound method could find optimal solutions for around 25 jobs in 8 minutes on a personal computer (Pentium 200 MHz).
Ng (2005) gives an extension of this model. Multiple cranes and non-interference constraints are included. This increases the complexity of the problem dramatically. Also

space constraints are needed to describe the position of a crane or job at a specific time interval. In this model RTG cranes can move between different blocks, but only in one lane. Again, the objective is to minimize the sum of completion times instead of minimizing the makespan. The solution method is via dynamic programming. A zone is assigned to every RTG crane based on the expected sum of completion times for each zone. Since determining the sum of completion times for a zone is NP-complete (Lenstra et al. (1977)), this is approximated via a greedy heuristic. The greedy heuristic takes the first job that can be completed as the next job. After obtaining a solution, a second step of the algorithm tries to improve this solution. One job is moved to another zone and a quick check is made to see if an improvement has been made.

Model of Guo and Huang (et al)

Level 1	Distribution of yard cranes to different lanes
Level 2	Time Partitioning into planning windows
	Space Partitioning into yard crane working zones
Level 3	Yard crane dispatching in individual zones

Figure 4.1: Hierarichal scheme of Guo and Huang

Guo et al. (2011); Guo and Huang (2012) have studied the *yard crane scheduling problem* thoroughly. Like Ng and Mak, they also try to minimize the sum of completion times. They propose a hierarchical model for yard crane operations (see Figure 4.1). At the top hierarchical level, they distribute the RTG cranes to different lanes. In the second level time partitioning and a space partitioning is done within the lane to create different planning windows and working zones. At the lowest level, the yard crane dispatching is done in individual zones. The top level is not included in their research. They assume this is done in advance and their procedure starts at level 2. Level 3 is a subroutine of level 2, and therefore has to be solved many times. So especially this dispatching within zones needs to be found very fast. They propose to use a lower bound function to bound the sum of completion times for a specific partitioning from below. This is used for prioritizing the search for partitionings and exclude partitionings that result in bad solutions. They use a modification of the same subroutine for the lowest level for this. In the next paragraphs the different levels are discussed.

Dispatching within zones (level 3) In Guo and Huang (2012), a fast simulation routine is used to find feasible solutions for this dispatching problem. In Guo et al. (2011),the problem is rewritten to a search tree and a A* search is used to find optimal solutions. A* search finds paths from a start node to leaf nodes in a search tree and is an extension of Dijkstra's shortest path algorithm. If the algorithm arrives at a node x during the process, it has calculated the shortest path to this node x from start and estimated the

path from x to leaf nodes. If the heuristic used to estimate the path to a leaf node is guarantied to find a lower bound to the optimal path, then this heuristic is called admissible. Let $d(x, y)$ denote the shortest path from x to y and $h(x)$ the estimated path to the leaf node. If $h(x) \leq d(x, y) + h(y)$ then h is called consistent. If h is admissible and consistent, then A* search is guaranteed to find an optimal solution. The heuristic used by Guo et al. (2011) calculates the handling time of all jobs to process plus the time to go to the first other job available. This is an admissible and consistent heuristic and therefore their algorithm will find optimal solutions. Generally A* search is used to find a shortest pad to all nodes from a starting node. The tree Guo et al use is build in such a way that the depth (nodes from root to leafs) is exactly the number of jobs for all leaves. Every node (parent) is split into new nodes (children) such that the union of the children are exactly the jobs that are not on the path from the root node to the parent node. A* has as disadvantage that it is breath first, so no solution is found in an early stage. Due to this, it also uses a lot of memory. They propose using Recursive Backtracking to improve the algorithm. RBA* gives a solution even if the entire search is not finished. RBA* works best when given a reasonable solution in an early stage. Prioritized search is used to realize this. This is done via job arrival times and via first reachable time. Computational results show that both prioritized RBA* and A* are reasonably fast, they produce results in up to 2s for 20 jobs in one hour for a block of 37 bays. They use a Pentium Core2 Quad CPU Q9450 and 3 GB RAM for their experiments. To find lower bounds during the partitioning process, they use A* search up to a specific number of nodes (% of total dept).

Space partitioning (level 2b) For the space partitioning, the algorithm starts by finding possible partitioning points. These points are between different blocks or at places where there are 8 bays adjacent where no jobs are within the planning period. This 8-bays-separation is used to assure that trucks can enter and leave beneath the RTG crane. R denotes the number of partitioning points and let m be the number of RTG cranes available. The partitioning algorithm needs to choose $m-1$ partitioning points out of R possible partitioning points. This results in $\binom{R}{m-1}$ possibilities. This number is highly dependent on R which results from the time partitioning. Since many partitioning points are necessary to find good schedules, each possible partitioning has to be evaluated very fast. They use a lower bound search that can be found very fast. A lower bound is evaluated for each partitioning and the partitionings are sorted based on their potential. Next full evaluation is used for the most promising partitioning. All partitionings with a larger lower bound than the optimum found thus far by the algorithm are discarded for further evaluation. In this way not all partitionings have to be fully evaluated but optimality can be assured (given the partitioning strategy).

Time partitioning (level 2a) Time partitioning is essential for a good space partitioning. A small time window results in many partition points which result in long computational time for the space partitioning. In addition, when the time windows are small there is

less opportunity to optimize the RTG crane dispatching. On the other hand, a large time window might result in too few partitioning points to find a feasible or balanced enough solution. Two algorithms are proposed for finding good time window partitioning. The first algorithm tries to break the time interval in two via two different options: 1) divide the time length in half, 2) divide the time length such that the number of jobs is equal. The algorithm keeps on dividing the time window until no better solution is found. The second algorithm will keep on dividing the time interval in equal time length until there is one job per crane. After that, the best solution is returned. The main difference is that algorithm 1 will terminate if a division in two time windows does not improve the solution while a division in more different time windows might.

Performance Simulation results show that the second time partitioning algorithm performs best in all examples that they have studied. The computation times for both algorithms are significantly longer than Ng (2005), but still computable within a few minutes. This is acceptable for them. Just space partitioning without time partitioning works worse than Ng (2005). Ng's algorithm is adjusted such that it includes a separation of 8 bays for the RTG crane. This might significantly decrease the job switching ability that was proposed as second stage by Ng.

MILP formulations of Li et al

Li et al. (2009) give different MILP representations for the *yard crane scheduling problem*. They use the model as proposed by Ng (2005),but rewrite the MILP formulations to reduce the number of binary variables. Instead of using tri-index binary variables they use two bi-index binary variables. They use a slightly different approaches in the modeling. They do not model the travel time of yard cranes according to the distances between slots. Cranes process one job in each time interval regardless of how far they have to travel. They restrict yard cranes to handle jobs that are no more than 8 slots apart in successive time intervals. This same separation constant is used to model the distance between two yard cranes during a time interval. When 2 yard cranes, 60 bays, 32 jobs and an 168 minute planning horizon are used, there are 1291 binary variables, 4813 continuous variables 84,212 single equations. The model of Ng (2005) uses 11,094 binary variables and 333,248 equations for a 2 hour planning horizon. Still, approximately 2 hours are needed to solve the problem to optimality on a Pentium 1.6GHz computer using ILOG CPLEX 9.0.

To reduce computation times, they develop a heuristic to find near optimal solutions. They note that in an optimal schedule the actual handling times are close to the due dates. So they suggest to only search for processing times close to the due dates. This reduces the search space to 398 binary variables, and 10 minutes of calculating an optimal solution (within the restricted model). Creating a rolling horizon program reduces the complexity even more. The approach of the last model is to schedule only the first few

jobs. All jobs in a short time window (eg 5 minutes) are scheduled, but only the jobs in the first 2 minutes are remembered. Then, two minutes are added to the planning window and a new schedule is calculate. Again the jobs scheduled in the first 2 minutes are added to the definite schedule which contains now the first 4 minutes. This procedure is repeated until all jobs are scheduled.

Genetic algorithms

More recent research is focused on genetic algorithms to find near optimal solutions to the *yard crane scheduling problem with interference constraints*. Mak and Sun (2009) use tabu search to enhance the performance of a genetic algorithm. They minimize the total makespan of the yard cranes instead of minimizing the waiting time of yard trucks. Javanshir and Seyedalizadeh Ganji (2010) used a genetic algorithm to minimize the completion time of all bays. This is the same as minimizing the makespan. He et al. (2010) use a hybrid genetic algorithm to minimize different weighted objectives. The first objective is to minimize the total delayed workload and the second objective is to minimize the number of block changing moves of yard cranes. Chang et al. (2011) use a genetic algorithm in combination with a rolling-horizon approach.

4.2.4 Other studies

Some other studies have been done in the subject of yard crane scheduling. They use different modeling assumptions than the studies introduced above. Wang and Yang (2013) give an MILP for the yard crane scheduling with interference constraints, including a stochastic component. The number of containers arriving at the yard during a shift for a certain ship is assumed stochastic. Their model assigns yard bits (a bit is part of a block) for storage and deploys yard cranes to handle the workload in those blocks.

Petering and Murty (2006) introduces an MILP formulation that assigns jobs to a specific time period without spatial constraints. In a second stage the distribution of jobs to the RTG cranes is made. This results in a simple MILP with few variables. In the second stage a dynamic programming procedure is used to divide the jobs into separate yard cranes zones. The assumption is made that subsequent jobs for a yard crane are not that far apart. The yard crane has only 4 minutes to perform gantry and job handling. On average, this should be enough.

Petering et al. (2009) gives a thorough analysis by simulation of yard crane dispatching decisions. They compare several simple heuristics in yard crane dispatching and analyze their influence on the performance of the whole terminal. Results strongly indicate that prioritizing retrieval moves over storage moves gives a superior system. Their main objective is to minimize vessel turn-around time, so delaying store moves does not directly influence the quay crane rate. They also conclude that avoiding deadlocks in 100% of the occasions in a very transparent way is preferable to look-ahead yard crane dispatching

systems. In their model, a yard crane can perform a retrieval job before the time it is scheduled by the quay crane. This results in a deadlock, since the yard truck might not be around when the yard crane finishes his job. They suggest such short planning periods that on average only 1.5 jobs are available for scheduling. Therefore, simple dispatching rules suffice.

Chapter 5

Models

In the previous chapter we have described different ways to model the *yard crane scheduling problem* as found in the litarture. In this chapter we introduce some modeling assumptions and three new models that will be compared in this thesis. The first one (Model1) uses the MILP formulation of Ng and Mak (2005), but we extend it to multiple yard cranes working in the same lane in a different way than Ng (2005). The second MILP formulation is an alteration of the formulation introduced by Li et al. (2009). Some constraints are adjusted to make it compatible with our model assumptions and to prevent crossing of cranes. As a third model we introduce a model based on the *traveling salesman problem*. The traveling salesman problem is studied extensively so it might be advantageous to use knowledge and MILP solvers of the traveling salesman problem for the *yard crane scheduling problem*. In this chapter we formulate the problem in a static form, i.e. for one fixed time interval, without uncertainty in the job ready times. In Chapter 7 we extend this in a rolling horizon fashion. In Chapter 6, a decomposition method is introduced for the first model.

5.1 Model assumptions

To model the *yard crane scheduling problem*, the following assumptions have been made. The first four assumptions are needed to define the model correctly, the other assumptions are more test case properties.

1. The available yard cranes have the same productivity, handling speed and gantry speed.
2. A specified number of yard cranes is available to work in the specific lane.
3. Jobs are planned including a ready time, handling time and location (bay number).

4. Yard cranes are numbered from 1 to m (number of cranes available) within a lane. The yard cranes are numbered increasingly from left to right.

5. A complete layout of the storage yard is known. The number of bays is known and bays are numbered increasingly from left to right. A block consists of 40 to 60 bays and are separated by a distance of 4 to 8 bays.

6. The distance between two jobs is measured in the number of bays the yard crane has to travel, so the spatial variable is measured in bays. The time it takes for a yard crane to gantry one bay is assumed to be constant (4 seconds).

7. The handling of a job is around 150 seconds, this includes the vertical movements of the spreader and the horizontal movement within the bay. The gantry time of cranes is not included.

8. The travel time of the yard cranes is linear in the number of bays. Other travel time functions can be easily inserted (such that they include acceleration for example), but they have to respect the triangle inequality.

9. Yard cranes are not allowed to cross each other. A separation space of 8 bays (SEP) is needed between to side by side yard cranes. This space is used by trucks to move from under the yard crane to the road at the side of the yard crane.

The jobs that are planned come in four different types. Load jobs and discharge jobs are jobs linked to the vessel. Load jobs are containers that have to be moved from the yard to the vessel. Regularly a sequence of containers is stored in the yard such that they can be loaded into the ship in sequence. For the yard this means that the jobs are planned at the same bay with 150 seconds in between, such that the yard crane can handle these jobs subsequently. In the case of discharge, a vessel is unloaded and the containers need to be stored in the yard. The internal trucks handling these jobs need to be back in time at the quay crane to get the next container. So load jobs and discharge jobs have high priority for the yard. Receivel and delivery jobs have a lower priority for the container terminal. These jobs come from external trucks and delay does not directly affect the throughput of the terminal. The different kind of jobs are illustrated in Figure 5.1.

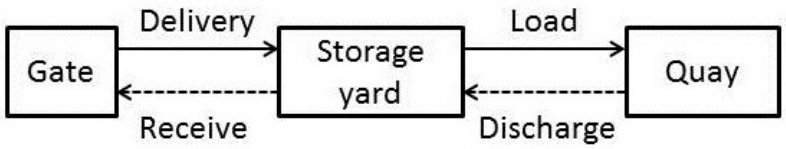


Figure 5.1: Different types of moves

Although yard cranes are technically able to process 24 jobs per hour in the same bay, in practice yard cranes perform around 10 jobs per hour depending on the amount of

work in a lane (experience TBA). The models that we introduce are used to solve the dispatching of several yard cranes (RTG cranes) within a lane. RTG cranes are able to change lanes, but this is not included in this model. Another procedure must distribute the number of cranes to a lane based on expected workload. This assumption is made to reduce the complexity of the problem. Changing lanes takes a lot of time and it blocks several roads during the process. Therefore it is preferable to have as few lane changes as possible. In the remainder of this thesis, an RTG crane will simply be referred to as "crane".

5.2 Model1

This first model uses the formulations of Ng and Mak (2005) for the dispatching of one crane in a zone. The model is extended to include several cranes and non-crossing constraints in a different manner than in Ng (2005). Instead of using tri-index binary variables to keep track of time, bi-index variables are used. This is done to significantly reduce the number of binary variables. To do this, we restrict the model by fixing zones for a planning period. This means that the lane is divided into m (the number of yard cranes) non-overlapping zones, one for each crane. This restriction might result in more delay, but it creates an opportunity for faster algorithms. The same restriction is imposed by Guo et al. (2011). The restriction to separate the zones is common at container terminals and is usually referred to as CHE (container-handling equipment) ranges. Nowadays this is usually done intuitively (without using an algorithm) by an employee of the container terminal. Calculating an optimal solution mathematically might enhance the performance.

5.2.1 Definitions

For a problem instance, the following constants are known:

n	the number of jobs	
m	the number of yard cranes	
θ	the number of bays	
r_i	the ready time (truck arrival time) of job i	$i = 1, \ldots, n$
h_i	the handling time of job i	$i = 1, \ldots, n$
b_i	the bay-number/position of job i	$i = 1, \ldots, n$
d_{ij}	distance of job i to job j (measured in time)	$i, j = 1, \ldots, n$
SEP	minimum separation space between two yard cranes	

The following will be the decision variables for the model:

$$
\begin{array}{lll}
t_i & \text{the completion time of job } i & i = 1,\ldots,n \\
x_{ij} & = \begin{cases} 1 & \text{if job } i \text{ precedes job } j \text{ in a zone;} \\ 0 & \text{otherwise.} \end{cases} & i,j = 1,\ldots,n \\
y_{ik} & = \begin{cases} 1 & \text{if job } i \text{ is in zone } k \text{ (handled by crane } k\text{);} \\ 0 & \text{otherwise.} \end{cases} & \begin{array}{l} i = 1,\ldots,n \\ k = 1,\ldots,m \end{array}
\end{array}
$$

Note that the variable $x_{ij} = 1$ even if job i does not come directly for job j. In problems as the traveling salesman problem, often $x_{ij} = 1$ if and only if job j comes directly after job i.

5.2.2 MILP formulation

The objective is to minimize the sum of delays of all jobs, i.e. $\min \sum_{i=1}^{n} \big(t_i - (r_i + h_i)\big)$. The delay is thus defined as the completion time of a job minus its ready time and handling time. It is therefore the actual completion time minus the first time the job could have been completed. In this model, M is a large enough number and is defined as: $M = \max_{i=1..n}(r) + n(\max_{i,j=1..n}(d) + \max_{i=1..n}(h))$. The function f describes whether job j is within a distance SEP of job i (or even at the right side of job i). The function f is defined as follows:

$$
f(b_i, b_j) = \begin{cases} 0 & b_j \geq b_i - SEP; \\ -m & \text{otherwise.} \end{cases} \qquad i,j = 1,\ldots,n
$$

Using the above mentioned variables, the yard crane scheduling problem can be formulated as:

$$
\text{Minimize:} \quad \sum_{i=1}^{n} \big(t_i - (r_i + h_i)\big) \tag{5.1}
$$

$$
\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1,2,\ldots,n \tag{5.2}
$$

$$
\sum_{l=1}^{m} y_{jl} \cdot l - \sum_{k=1}^{m} y_{ik} \cdot k \geq f(b_i, b_j) \qquad i,j = 1,\ldots,n; i \neq j \tag{5.3}
$$

$$
t_i \geq r_i + h_i \qquad i = 1,2,\ldots,n \tag{5.4}
$$

$$
y_{ik} + y_{jk} - 1 \leq x_{ij} + x_{ji} \qquad i,j = 1,\ldots,n; i \neq j; k = 1,\ldots,m \tag{5.5}
$$

$$
t_j - t_i - M x_{ij} \geq d_{ij} + h_j - M \qquad i,j = 1,\ldots,n; i \neq j \tag{5.6}
$$

$$
x_{ij}, y_{ik} \in \{0,1\} \qquad i,j = 1,\ldots,n; i \neq j; k = 1,\ldots,m \tag{5.7}
$$

The model uses $n(n-1) + nm$ binary decision variables and n continuous decision variables. There are $(2+m)n^2 - mn$ constraints. Constraint (5.2) ensures that each job

is assigned to exactly one zone/crane. Constraint (5.3) is the zone overlapping constraint and ensures that a job in a lower zone cannot be at a higher bay number than a job in a higher zone. For example, set $b_j \geq b_i - SEP$ such that $f(b_i, b_j) = 0$. If job j is handled by crane k, then Constraint (5.3) prohibits cranes $l = k+1, k+2, \ldots, m$ to perform job i. If $b_j < b_i - SEP$, then $f(b_i, b_j) = -m$ and the constraint is always (for all possible values of y_{jl} and y_{ik}) satisfied. For implementation, one can even decide to leave the constraints where $b_j < b_i - SEP$ out of the formulation. Note that f is a non-linear function, but its variables are the bay numbers and not the decision variables of the MILP problem. Therefore this MILP formulation is still a linear problem since the (scalar) values can be determined a priori. Constraint (5.4) demands that a job cannot be completed earlier than its ready time plus handling time. From Constraint (5.5) it follows that if two jobs are in the same zone, then either job i precedes job j in the cycle or job j precedes job i. In (5.6) the relationship between the completion time of two jobs in the same cycle is given. Finally, (5.7) are the binary constraints.

The constraints with $i = j$ are omitted since they do not represent real constraints. It would also result in infeasibility since $x_{ii} = 0$ conflicts with (5.5) while $x_{ii} = 1$ conflicts with (5.6). Note that $t_i > 0$ follows directly from Constraint (5.4). Constraint (5.5) gives us $y_{ik} = 1 \wedge y_{jk} = 1 \rightarrow x_{ij} = 1 \vee x_{ji} = 1$, but not the other way around. This is not a problem since we minimize t_j and in Constraint (5.6), x_{ij} will be chosen $x_{ij} = 0$ if possible and necessary for an improved solution.

5.2.3 LP-relaxation

The runtime of algorithms for an MILP is often very dependent on the quality of the associated LP relaxation. For a good MILP formulation it is required that the solution space of its LP relaxation is close to the convex hull of the solution space of the MILP (Bertsimas and Tsitsiklis (1997)). For our model, the LP relaxation optimum is always zero and can often be a lot smaller than the optimum of the MILP. Since the optimal value of the LP-relaxation is zero, the integrality gap is not defined. The difference between the optimal value of the LP relaxation and the MILP problem itself means that a large relative integrality gap exists for almost all problem instances. This suggest that Model1 is a "bad" formulation. The "big-M" constraint (5.6) is a problem for the LP-relaxation.

Because the x variables are no longer restricted to 0 or 1, Constraint (5.6) becomes inactive when $x_{ij} < 1$. The x variables are further only restricted by (5.5). Choosing both x_{ij} and x_{ji} equal to a half can result in all inactive constraints of type (5.6). To reduce this problem, M is chosen as small as possible. But since the constraints must always hold, it is not possible to choose M small enough to be useful for the LP relaxation. Another problem in the LP relaxation is that it is possible to choose $y_{i1} = y_{i2} = \frac{1}{2}$ for all i. This corresponds to letting two cranes perform all jobs half. This satisfies Constraints (5.2) and (5.3) in any case (provided $m \geq 2$), but ensure that Constraint (5.5) stays non-restrictive. The left hand side becomes zero, so there

is no restriction for the x variables.This is what actually happens when we solve the LP-relaxation. As a consequence, all constraints of type (5.6) become inactive and $t_i = r_i + h_i$. This results in zero delay. Zone dividing and restrictions between jobs are completely discarded.

Example: Assume we have a problem with 2 cranes and 5 jobs. If we relax the binary constraints, we can choose $y_{i1} = y_{i2} = \frac{1}{2}$ for $i = 1, \ldots, 5$ and $x_{ij} = 0$ for $i, j = 1, \ldots, n; i \neq j$. These values satisfy constraints (5.2), (5.3), and (5.5). In the right-hand side of (5.6), the big M ensures that the constraints becomes inactive. The only restriction on the t-variables (and thus the objective value) is constraint (5.4). Therefore we can choose $t_i = r_i + h_i$ for $i = 1, \ldots, 5$. This results in $\sum_{i=1}^{5} t_i = \sum_{i=1}^{5}(r_i + h_i)$ which corresponds to zero delay.

Within this model it is difficult to adjust the constraints such that the LP-relaxation becomes stronger. A possible adjustment would be to switch to a quadratic constraint and use quadratic programming (e.g.: $x_{ij}(t_j - t_i - d_{ij} - h_j) \geq 0$). Because quadratic programming is more difficult to solve than linear programming, we continue to use linear constraints.

5.2.4 Alternatives

Some alternative formulations for the above proposed model can be constructed by making a number of small adjustments. These alternatives are described briefly below. The focus of the adjustments is mainly on reducing the solution space. However, they usually do not result in a smaller relative integrality gap. In addition, no reduction in computation times for Gurobi is identified. A possible adjustment that did not result in longer computation time for Gurobi is an adjustment for the constraints of type (5.3). In the Constraints (5.3) the non-linear function f forms some sort of a "big M" constraint when $b_j < b_i - SEP$. This can be avoided by replacing (5.3) by (5.8):

$$\sum_{\kappa=1}^{k} \left(y_{i\kappa} - y_{j\kappa}\right) \geq f'(b_i, b_j) \qquad i, j = 1, 2, \ldots, n \quad k = 1, 2, \ldots, m \tag{5.8}$$

$$f'(b_i, b_j) = \begin{cases} 0 & b_j \geq b_i - SEP; \\ -1 & \text{otherwise.} \end{cases} \qquad i, j = 1, \ldots, n$$

Assume that $b_i < b_j$. If job i is in the same zone as job j, the left hand side is 0 for all k. However, if job i is handled by crane k_1 and job j is handled by crane k_2 and $k_1 < k_2$, the left hand side becomes $+1$ for $k = k_1$. This is allowed by the constraint and also by the physical constraint. If on the other hand $k_2 < k_1$, then $\sum_{\kappa=1}^{k_2} y_{i\kappa} - y_{j\kappa} = -1$ which violates the constraint as required. If job i is left of job j ($b_i < b_j$), job j cannot be handled by a crane left of the crane which handels job i.

In the Constraints (5.8) the "big M" is reduced from $-m$ to -1. This comes, however, at the cost of adding extra constraints. (5.8) consists of $n^2 m$ constraints, while (5.3) gives

us n^2 constraints. It is a trade-off between solution space and number of constraints. The difference in runtime for Gurobi between using (5.8) and (5.3) was minimal. Although (5.8) does reduce the solution space, it does not reduce the integrality gap. Therefore we stick to the smaller number of constraints by using (5.3).

Another possible adjustment is to introduce the extra constraints (5.9).

$$x_{ij} + x_{ji} \leq y_{ik} - y_{jk} + 1 \quad i, j = 1, \ldots, n; k = 1, \ldots, m \tag{5.9}$$

These constraints forbid x_{ij} and x_{ji} to be both 1 when job i and job j are in the same zone. Additionally, it ensures that $x_{ij} = x_{ji} = 0$ when job i and job j are in different zones. The addition of these constraints result in longer runtime of Gurobi. Therefore the addition seems inappropriate. The increase in runtime is, however, small and we will see in Chapter 6 that in combination with a Benders decomposition, it might be preferable to add these constraints. Finally, we could also introduce new variables z that describe direct precedents of jobs. This is a more generally used type of binary variable in other applications. We have found, however, that the addition of these variables resulted in much longer computation times for Gurobi. The introduction of more binary variables is worked out in Appendix A.

5.3 Model2

Li et al. (2009) present a model that is also based on the model of Ng and Mak. Their model uses a time-index to identify the sequence in which jobs need to be performed. This has the advantage that no "big M" constraint is needed as in Model1. This model also does not fix zones for the cranes. This has the advantage that yard cranes can work in the same bay at different times. A disadvantage is that the time is discretized to 3 minutes. Yard cranes can only perform one job per 3 minutes. In two subsequent time intervals a yard crane can only perform 2 jobs if they are at most 8 bays apart. This reduces the flexibility of yard cranes, because it takes three time intervals (9 minutes) for a yard crane to perform 2 jobs that are 10 bays apart, whereas 5 minutes and 40 seconds is possible in reality.

We present the model introduced by Li et al. (2009) with some adjustments. These adjustments are done to make it compatible with our problem definition. This model uses time intervals to determine the location of a yard crane at a certain moment. Just like Model1, this model uses bi-index binary variables. Instead of using variables to succesive jobs, their variables describe crane to job assignment or job to time interval assignment. In Model2, a crane can perform 1 move in each time interval. It uses a time interval (of 3 minutes) to travel to the next location and pick up a container. It can only perform a job in succeeding time intervals when their location is at most 8 bays apart. Yard cranes cannot be at the same bay at the same time. However, they can be at the same bay at different times. Model1 fixes zones for the entire planning period, while Model2 does not fix zones and allows crane $k+1$ to be at bay 6 when crane k was at bay

6 in an earlier time interval. As in the model proposed by Li et al. (2009) cranes have to be at least 8 bays apart in the same time interval. The difference between storage moves and retrieval moves is omitted. Jobs can only be performed after their ready time. This corresponds to the notion of storage moves in Li et al. (2009).

5.3.1 Definitions

The following constants are assumed given:

n	the number of jobs
m	the number of yard cranes
θ	the number of bays
r_i	requested handle interval of job i
b_i	the bay-number/position of job i
T	the number of time intervals
SEP	the minimal number of bays between yard cranes

The following will be the binary decision variables for the model:

$$y_{ik} = \begin{cases} 1 & \text{if job } i \text{ is done by crane } k; \\ 0 & \text{otherwise.} \end{cases}$$
$$x_{it} = \begin{cases} 1 & \text{if job } i \text{ is handled at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

The model also includes continuous decision variables z_{ikt} for $i = 1, \ldots, n$; $k = 1, \ldots, m$; t=1...,T. These variables are continuous, but will be 0 or 1 for every feasible solution. The z-variables are linked to the binary x and y-variables by constraints in the model. They therefore represent the following decisions.

$$z_{ikt} = \begin{cases} 1 & \text{if job } i \text{ is handled by crane } k \text{ at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

5.3.2 MILP formulation

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{t=1}^{T} t \cdot x_{it} \tag{5.10}$$

$$\sum_{t=1}^{T} x_{it} = 1 \qquad i = 1,\dots,n \tag{5.11}$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1,\dots,n \tag{5.12}$$

$$\sum_{t=1}^{T} t x_{it} \geq r_i \qquad i = 1,\dots,n \tag{5.13}$$

$$\sum_{i=1}^{n} x_{it} \leq m \qquad t = 1,\dots,T \tag{5.14}$$

$$z_{ikt} + 1 \geq x_{it} + y_{ik} \qquad i = 1,\dots,n; k = 1,\dots,m; t = 1,\dots,T \tag{5.15}$$

$$z_{ikt} \leq y_{ik} \qquad i = 1,\dots,n; k = 1,\dots,m; t = 1,\dots,T \tag{5.16}$$

$$z_{ikt} \leq x_{it} \qquad i = 1,\dots,n; k = 1,\dots,m; t = 1,\dots,T \tag{5.17}$$

$$\sum_{j \in B_i^1} z_{j,k+1,t} \leq 1 - z_{ikt} \qquad i = 1,\dots,n; k = 1,\dots,m; t = 1,\dots,T \tag{5.18}$$

$$\sum_{j \in B_i^2} z_{j,k,t+1} \leq 1 - z_{ikt} \qquad i = 1,\dots,n; k = 1,\dots,m; t = 1,\dots,T \tag{5.19}$$

$$\sum_{i=1}^{n} z_{ikt} \leq 1 \qquad k = 1,\dots,m; t = 1,\dots,T \tag{5.20}$$

$$x_{it}, y_{ik} \in \{0,1\} \qquad i = 1,\dots,n; k = 1,\dots,m; t = 1,\dots,T \tag{5.21}$$

$$B_i^1 := \{j : j \in \{1,\dots,n\}, b_j \leq b_i + SEP\}$$
$$B_i^2 := \{j : j \in \{1,\dots,n\}, |b_j - b_i| > 8\}$$

The first set of constraints (5.11) ensure that every job is scheduled at exactly one time interval. Constraints (5.12) make sure that every job is assigned to a crane. Constraints (5.13) are used to make sure that a job is not performed before it is available. Constraints (5.14) limit the number of jobs scheduled at a certain time interval to the number of cranes. Constraints (5.15), (5.16) and (5.17) define the variables z. Constraints (5.18) functions as non-crossing constraint for the cranes. Crane $k+1$ cannot perform a job with a smaller bay number than the bay number of the job performed by crane k minus SEP at the same time interval. For example, assume that crane k performs job i at bay 7 at time t. Job j at bay 9 cannot be performed at time t by crane $k+1$. Constraints (5.19) limit the number of bays a crane can gantry to 8 between succeeding jobs. A crane

cannot perform more than one job in each time interval. This is modeled by Constraints (5.20). The sets B_i^1 and B_j^2 are used to describe for which j the decision variables z are present.

The model as presented by Li et al. (2009) does not prevent cranes from colliding. In their model, the left-hand side of (5.18) is summed over all j such that $b_j < b_i - SEP$. They restrict cranes from crossing each other only when they perform a job. When they use a time interval for gantrying, this crossing is not prohibited. This can result in optimal schedules that are not feasible. For example, let $b = [1, 11, 21, 31]$ and $r = [1, 4, 2, 3]$. Model2 as presented above will let Crane 1 handle the first and fourth job at $t = 1$ and $t = 3$. Crane 2 will handle the second and the third job at $t = 4$ and $t = 2$. Crane 1 must cross Crane 2 while gantrying to job 4 at $b = 31$. Letting crane 1 perform jobs 1 and 2 and crane 2 jobs 3 and 4, will result in a delay equivalent to one for crane 2 (see Figure 5.2. However, that is the best solution that is feasible in practice.

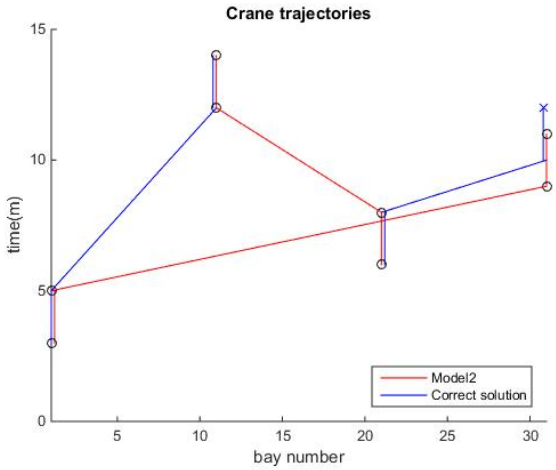


Figure 5.2: Crossing of cranes with Constraint (5.18)

We adjust this by changing the left-hand side of (5.18). If the left-hand side sums over all time intervals instead of just the one considered in the constraint, zones are fixed for the entire planning period just as in Model1. The cranes lose the ability to perform jobs at the same bay at different time intervals, but collision is avoided. Instead of summing over all possible t, we will sum over $t-1, t, t+1$. This is enough to avoid crossing of cranes. We also sum over all cranes k' for $k' > k$ instead of using only $k+1$. Otherwise it might happen that if crane $k+1$ is idle, crane $k+2$ can be scheduled to perform jobs at the left side of crane k. At the right-hand side of (5.18) we multiply by $3(m-1)$ to ensure that when $z_{ikt} = 0$, the constraint remains inactive. Instead of (5.18) we

obtain:

$$\sum_{j \in B_i^1} \sum_{\kappa=k+1}^{m} \sum_{\tau=t-1}^{t+1} z_{j,k+1,\tau} \leq 3(m-1)(1-z_{ikt}) \quad i = 1,\ldots,n; k = 1,\ldots,m; t = 1,\ldots,T \tag{5.22}$$

As pointed out in Li et al. (2009) we use the "heuristic" constraints that jobs cannot be scheduled later than T_{lim} time intervals beyond their planned time interval. This reduces the number of binary variables and the number of constraints. T_{lim} is set to 8 as suggested by Li et al. (2009). This means that a lot of binary variables are fixed to zero and constraints are only present for some t. For example, for the constraints (5.22) this means that the constraints are only present for $t = r(i),\ldots,r(i)+8$. The same holds for constraints (5.15),(5.16),(5.17), and (5.19). The constraint (5.23) is added to the problem to set some decision variables to zero. Since all decision variables are either zero or one, Constraint (5.23) sets all variables that are present in the equation to zero.

$$\sum_{i=1}^{n} \sum_{k=1}^{m} \left(\sum_{t=1}^{r(i)-1} (x_{it} + z_{ikt}) + \sum_{t=r(i)+T_{lim}+1}^{T} (x_{it} + z_{ikt}) \right) = 0 \tag{5.23}$$

5.4 Model3

For Model1 we used the model of Ng and Mak (2005) for one crane as a starting point. The model was extended via assignment constraints to describe multiple cranes. A different way to model the problem is via a multiple traveling salesman problem (mTSP). We will extend the model proposed in Burger (2014) for the mTSP by adding non-crossing constraints and job ready times. Fictive depots are introduced in this model from where cranes/salesman depart from and will arrive to. We will refer to this model as Model3. Although this model shows similarities with a traveling salesman problem, the performance of Gurobi (compared to Model1) is not improved by modeling it in this manner.

We introduce a fictive depot as start and finish point for each yard crane. This allows us to model this problem as a fixed-destination multiple traveling salesman problem (mTSP). In the mTSP there are multiple depots each having one salesman. The problem contains a set of cities that have to be visited exactly once by exactly one salesman. A feasible solution consists of routes for each salesman such that every city is visited exactly once. So the set of cities must be divided into disjoint subsets, one for each salesman. The objective is to minimize total traveling distance of all salesmen. For our *yard crane scheduling problem*, the depots are modeled as dummy jobs. For these jobs the bay number, ready time and handling time are set to zero. Also the traveling time from or to bay zero is set to zero for all bays. To model the mTSP we use constraints as given in Burger (2014). In addition to these constraints, the model includes extra

constraints (5.29) to model the non-crossing of cranes. These constraints are not in the regular mTSP problem. Also the constraints (5.30), demanding that jobs cannot be finished before the sum of their ready and handling time is not in the mTSP model. Therefore we also add these constraints, which are a variation on time-windows for the mTSP. A TSP with time-windows is a TSP where every city must be visited within a certain time-interval. For the non-crossing constraints we can make use of the "node currents". They give every node in a tour of a specific salesman (yard crane in our case) a unique number which can be used to model the non-crossing constraints. The subtour elimination constraints can be omitted, because we can use the completion time of the jobs for that purpose. Note that there is a strong resemblance between constraints (5.31) and the subtour elimination constraints MTZ introduced by Miller et al. (1960).

5.4.1 Definitions

The following constants are assumed given:

n	the number of jobs	
m	the number of yard cranes	
θ	the number of bays	
r_i	the ready time (truck arrival time) of job i	$i = 1, \ldots, n$
h_i	the handling time of job i	$i = 1, \ldots, n$
b_i	the bay-number/position of job i	$i = 1, \ldots, n$
w_i	the weight of job i	$i = 1, \ldots, n$
d_{ij}	distance of job i to job j (measured in time)	$i, j = 1, \ldots, n$
SEP	minimum separation space between two yard cranes	

The following will be the (decision) variables for the model:

$$z_{ij} = \begin{cases} 1 & \text{if job } j \text{ follows job } i \text{ directly for a crane;} \\ 0 & \text{otherwise.} \end{cases}$$

k_i depot/crane number of job i

t_i completion time of job i

Note that in this case the variable z describes the direct successor in contrast to the variable x in Model1. In Model1 x was used to denote that i comes before j in a cycle. We use the same big number M and function f as in Model1.

5.4.2 MILP formulation

$$\text{Minimize:} \quad \sum_{i=1}^{n} \big(t_i - (r_i + h_i)\big) \tag{5.24}$$

$$\sum_{j=1}^{m+n} z_{hj} = 1 \qquad h = 1, \ldots, m+n \tag{5.25}$$

$$\sum_{i=1}^{m+n} z_{ih} = 1 \qquad h = 1, \ldots, m+n \tag{5.26}$$

$$k_d = d \qquad d = 1, .., m \tag{5.27}$$

$$k_i - k_j + (m-1) z_{ij} \leq m - 1 \qquad i, j = 1, \ldots, m+n \tag{5.28}$$

$$k_j - k_i \geq f(b_i, b_j) \qquad i, j = m+1, \ldots, m+n \tag{5.29}$$

$$t_i \geq r_i + h_i \qquad i, j = 1, \ldots, m+n \tag{5.30}$$

$$t_j - t_i - M z_{ij} \geq d_{ij} + h_j - M \qquad j = m+1, \ldots, m+n \tag{5.31}$$

$$z_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, m+n \tag{5.32}$$

5.5 Analytical comparison

In this section we compare the models on properties as the number of constraints, variables and the quality of the LP-relaxation. The runtime of an algorithm used to solve the above models is hard to predict, since it depends on a lot of different properties. We solve the models using Gurobi which makes use of a branch and bound algorithm to solve mixed integer linear programming problems. Branching is applied to the binary variables via different search methods. Therefore the number of binary variables has a large influence on the number of branching possibilities. The number of constraints is of influence on the solution space. More constraints may lead to a smaller solution space. The size and shape of the solution space has influence on the search time of an algorithm. More constraints restrict the solution space, but also more verification of solutions need to be done. The LP-space is also important for the performance of algorithms. If the convex hull of an MILP is close to its LP relaxation, branch and bound algorithms can typically find an optimal solution faster.

5.5.1 Variables and constraints

All three models use different numbers of variables and constraints. These numbers are summarized in Table 5.1. We make a distinction between binary variables and

continuous variables. The influence on the runtime of the binary variables is much larger than that of the continuous variables. In the number of constraints, the binary constraints are excluded. Model2 Heur denotes the model as presented in Section 5.3 where the heuristic is implemented. Model2 denotes the model without the heuristic on handling intervals.

Model	Binary variables	Continuous variables	Constraints
Model1	$n(n-1+m)$	n	$(2+m)n^2 - mn$
Model2	$nm + nT$	nmT	$3n + (m+1)T + 5nmT$
Model2 Heur	$nm + n(T_{lim}+1)$	$nm(T_{lim}+1)$	$3n + (m+1)T + 5nm(T_{lim}+1) + 1$
Model3	$n(n-1+2m)$	$2(n+m)$	$3n^2 + 2nm + 3n + m^2 + 4m$

Table 5.1: Number of variables dependent on n,m and T

Since these amounts depend differently on the number of jobs, cranes and time intervals, we will fix these parameters. For comparison we look at the problem of scheduling $n = 20$ jobs for $m = 2$ cranes. The planning window is one hour, so $T = 20 + 8 = 28$ time intervals are used for Model2 (one time interval takes 3 minutes). T_{lim} (additional time intervals for delayed jobs) is fixed to 8 as proposed by Li et al. (2009). The number of variables and constraints for these specific values is given in Table 5.2.

Model	Binary variables	Continuous variables	Constraints
Model1	420	20	1560
Model2	600	1120	5744
Model2 Heur	220	360	1945
Model3	460	44	1352

Table 5.2: Number of variables and constraints for $n = 20, m = 2$ and $T = 28$

As is illustrated in Table 5.2, Model2 uses more binary variables than Model1 and Model3 for small problem instances. It also uses more continuous variables and more constraints. If we assume that the number of jobs scheduled per crane per hour is constant, then changing the planning window will not have a large impact on the ratio of binary variables between Model1 and Model2. If we increase the number of cranes (and thus the number of jobs), the ratio of binary variables will change in favor of Model2. The opposite is true for the number of continuous variables. Implementing the heuristic for Model2 has a strong influence on the number of binary variables and the number of constraints. Model2 Heur uses clearly less binary variables and constraints than the other models. Increasing the planning window or number of cranes will have the most impact on Model2 and the smallest impact on Model1 for continuous variables. The same holds for the number of constraints. This is illustrated in Table 5.3.

Model	Binary variables	Continuous variables	Constraints
Model1	2.700	50	17.250
Model2	1.650	7.000	35.318
Model2 Heur	700	2.250	11.569
Model3	2.950	110	8.195

Table 5.3: Number of variables and constraints for $n = 50, m = 5$ and $T = 28$

5.5.2 LP-relaxation

As described in Section 5.2 the LP-relaxation of Model1 leads to large relative integrality gaps. The optimal value of the LP-relaxation is always zero. This might lead to long computing times for Gurobi. The same problems occur in Model3. The complicating "big-M" constraints result in a large LP solution space. Model2 does not have such a "big-M" constraint. The objective of the LP relaxation of Model2 is not always zero although it is in general much smaller than the solution of the MILP. Simulating an experiment with 2 cranes and 20 jobs in 1 hour results in an average delay of 526.7 seconds for the LP relaxation and 1337.4 seconds for the MILP.

5.5.3 Conclusion

Based on the number of binary variables and the LP relaxation, Model2 seems a better formulation than Model1. Both models reflect different modeling decisions which lead to different optimal values. In Chapter 9 this is compared in detail. Although the above analytical results show a clear favor for Model2 for the expected runtime, experimental results show a less clear distinction. In the next chapter Benders decomposition is introduced. With the aid of this decomposition method the characteristics (number of constraints and variables) for Model1 can be improved a lot. By splitting in multiple subproblems, the overall number of binary variables is greatly reduced for Model1. The above calculations on variables and constraints are even worse for Model3. On the other hand, the TSP and mTSP are extensively studied at the moment. It might be possible to apply these results to construct better algorithms instead of directly solving with Gurobi for this model. Given the very discouraging results by using Gurobi (see Chapter 9), it might be hard to construct algorithms with shorter computing times compared to solving Model1. Advanced solvers for TSP or mTSP are not well suited for time constraints and cannot include other constraints such as our non-crossing constraints.

Chapter 6

Benders decomposition

Inspired by the ideas of Cao et al. (2010), we think that Benders decomposition might improve the runtime of Gurobi on the presented models. Benders decomposition was first introduced by Benders (1962). It divides the problem in a master problem and one or several subproblems, which are sometimes referred to as slave problems. Benders decomposition may be useful when certain variables are complicated while others are easy (Castillo et al. (2006), Chapter 3). This is in general the case for mixed integer linear programs since binary or integer variables are complicating while continuous variables are relatively easy. This suggests to split the problem in a master problem containing the binary variables (or generally the integer variables) and a subproblem containing the continuous variables. While iterating between the master problem and the subproblem, the algorithm "learns from past mistakes" (Hooker and Ottosson (2003)). This means that a feasible solution for the master problem is cut out if it results in a bad optimal value for the subproblem. This cut is usually based on the dual values. Since the subproblem consists of only continuous variables, it is a linear program and therefore dual values are easily computed.

In the first section we will present the general idea of the Benders decomposition. Different Benders decomposition methods are known. We will introduce a variant (logic-based Benders decomposition) thereafter. In the third section we will work out this logic-based Benders decomposition method for our *yard scheduling problem*. More Benders decomposition methods are described in Appendix B. Classical Benders decomposition methods as well as Benders decomposition methods designed for MILP problems are used on the presented models. These methods did not improve the computation times for this *yard crane scheduling problem*. We try to identify why these methods did not work well. Since the logic-based Benders decomposition method did reduce computation times for some problem instances, this method is presented within this chapter.

6.1 The Benders decomposition method

Benders decomposition exploits a block structure in the linear LP/MILP problem formulation to decompose the problem into several smaller problems. Subproblems need to be solved fast and lead to new constraints for the master problem. In a typical iteration, first the master problem is solved. This fixes some values that are used as input for the subproblems. The subproblems are solved using the values of the variables in the master problem. These results are translated to a Benders cut (defined later) which should restrict the MP for the next iteration (delayed row-generation). The objective function might depend on the variables in the subproblem, the variables in the master problem or both. Depending on this, different cut-generation methods are needed. Assume we have the following general problem, where y is a vector of binary variables and $x_1, \ldots, x_k$ vectors of continuous variables:

$$
\begin{array}{lrcrcrcrcrcl}
\text{minimize} & \mathbf{c}^T\mathbf{y} & + & \mathbf{f}_1^T\mathbf{x}_1 & + & \mathbf{f}_2^T\mathbf{x}_2 & + & \cdots & + & \mathbf{f}_k^T\mathbf{x}_k & & \\
\text{subject to} & \mathbf{A}\mathbf{y} & & & & & & & & & = & \mathbf{b} \\
& \mathbf{B}_1\mathbf{y} & + & \mathbf{D}_1\mathbf{x}_1 & & & & & & & = & \mathbf{d}_1 \\
& \mathbf{B}_2\mathbf{y} & & & + & \mathbf{D}_2\mathbf{x}_2 & & & & & = & \mathbf{d}_2 \\
& \vdots & & & & & & \ddots & & & \vdots & \\
& \mathbf{B}_k\mathbf{y} & & & & & & & + & \mathbf{D}_k\mathbf{x}_k & = & \mathbf{d}_k
\end{array}
$$

$$
\mathbf{y} \in \{0,1\} \quad \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \geq 0
$$

In this problem $\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_k, \mathbf{D}_1, \ldots, \mathbf{D}_k$ are constraint matrices, $\mathbf{c}, \mathbf{f}_1, \ldots, \mathbf{f}_k$ cost vectors and $\mathbf{b}, \mathbf{d}_1, \ldots, \mathbf{d}_k$ right hand side vectors. We define vectors as column vectors and use the notation $\mathbf{c}^T$ for the transpose of a vector $\mathbf{c}$. We use boldface lowercase letters for vectors and bold uppercase letters for matrices. This problem can be decomposed into a master problem and k different subproblems. The formulation of the subproblems and the resulting Benders cuts differ slightly in the literature. This formulation follows (among others) Bertsimas and Tsitsiklis (1997). The Benders cut is based on the dual objective of the subproblems. The master problem is given as:

$$
\begin{array}{ll}
\text{minimize} & \mathbf{c}^T\mathbf{y} + \alpha_1 + \alpha_2 + \cdots + \alpha_k \\
\text{subject to} & \mathbf{A}\mathbf{y} = \mathbf{b} \\
& \textbf{Benders cuts} \\
& \mathbf{y} \in \{0,1\}
\end{array}
$$

We have for $i = 1, \ldots, k$ the following subproblems where $\bar{\mathbf{y}}$ is the current solution of the master problem.

$$
\begin{array}{lrcl}
\text{minimize} & \mathbf{f}_i^T\mathbf{x}_i & & \\
\text{subject to} & \mathbf{D}_i\mathbf{x}_i & = & \mathbf{d}_i - \mathbf{B}_i\bar{\mathbf{y}} \\
& \mathbf{x}_i & \geq & 0
\end{array}
$$

This subproblem is a linear problem, since the x-variables are continuous. This allows us to construct the dual of the above formulated problem:

$$\begin{array}{ll} \text{maximize} & \mathbf{u}_i^T(\mathbf{d}_i - \mathbf{B}_i\bar{\mathbf{y}}) \\ \text{subject to} & \mathbf{u}_i^T\mathbf{D}_i \leq \mathbf{f}_i \\ & \mathbf{u}_i \quad \text{free} \end{array}$$

After solving the subproblems, we update the master problem with additional Benders cuts. These take the following form:

$$\alpha_i \geq \mathbf{p}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y}) \tag{6.1}$$

where $\mathbf{p}_i$ is the vector of dual values corresponding to an optimal solution of subproblem i. Since $\mathbf{p}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y})$ corresponds to the objective of the dual solution, strong duality guarantees that α_i is chosen at least as large as the optimal value of the subproblem when $\mathbf{y} = \bar{\mathbf{y}}$. If the subproblem is infeasible, we add the cut:

$$0 \geq \mathbf{w}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y}) \tag{6.2}$$

where $\mathbf{w}_i$ is an unbounded ray for the dual problem. We will not formally define an unbounded ray because it would also need the notion of other linear algebra concepts. A formal definition could for example be found in Bertsimas and Tsitsiklis (1997). Informally an unbounded ray is a direction in the solution space that will always lead to more feasible solutions. Multiplying an unbounded ray with a constant will still lead to another feasible solution, no matter how large this constant is. Proceeding via this unbounded ray will lead to an improved objective function which makes the optimization problem unbounded.

Note that the assumption that $\mathbf{f} \geq \mathbf{0}$ is sufficient for $\mathbf{u} = \mathbf{0}$ to be a feasible solution of the dual. Therefore the dual has either an optimal solution equal to the primal optimal solution or the dual has an unbounded ray corresponding to an infeasible primal. If in the master problem $\mathbf{y}$ is chosen such that $\mathbf{w}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y}) \geq 0$, then the dual of the subproblem becomes unbounded. This means that the primal of the subproblem is infeasible and therefore the complete problem is infeasible. This justifies the use of cut (6.2).

It is also possible to use the primal objective for the Benders cut. If the dual has a finite optimal value, strong duality guaranties that this is essentially the same. If the primal is unbounded artificial variables are added to construct a Benders cut. This process is somewhat different compared to the case presented above. Both methods are regular in literature. See Appendix B for a more detailed description.

6.2 Logic-based Benders decomposition

The classical Benders decomposition as presented above did not lead to good results. The problem was split between continuous and binary variables to exploit the advantages

of a linear subproblem. However, for our yard crane scheduling problem it is more advantageous to split between the binary variables. A part of the binary variables is in the master problem while the other part is in the subproblem (the subproblem also includes the continuous variables). For a splitting between different binary (or integer) variables different cuts need to be defined because we cannot use the strong duality results from LP subproblems. Hooker and Ottosson (2003) introduces logic-based Benders cuts that can be used for a wide class of constrained programming problems. It is well applicable for MILP problems where the subproblem is also an MILP. The idea of the Benders decomposition is the same, but the duals and Benders cuts are constructed for all kinds of constrained programming problems. The classical Benders decomposition is a special case of this logic-based Benders decomposition. For an LP subproblem, the classical dual is the same as the interference dual. Also, the classical Benders cuts are valid Benders cuts for LP subproblems.

Consider the following general optimization problem:

$$\begin{array}{ll} \text{minimize} & f(x,y) \\ \text{subject to} & (x,y) \in S \\ & x \in D_x, y \in D_y \end{array}$$

D_x is the domain of x, D_y is the domain of y and S is the set of feasible solutions. D_x and D_y could for example be the constraints that fix x, y to be binary and S a set of linear constraints. As in the previous section we decompose the problem between x and y variables. We get the following subproblem dependent on the current trial solution $\bar{y}^\kappa$.

$$\begin{array}{ll} \text{minimize} & f(x,\bar{y}^\kappa) \\ \text{subject to} & (x,\bar{y}^\kappa) \in S \\ & x \in D_x \end{array}$$

The next step is to identify a dual problem. In this case we cannot use the general notion of a dual from LP problems. Therefore we use the concept of an interference dual. An interference dual is defined as follows:

$$\begin{array}{l} \text{maximize } \beta^\kappa \\ \text{s.t. } (x,\bar{y}^\kappa) \in S \xrightarrow{D_x} f(x,\bar{y}^\kappa) \geq \beta^\kappa \end{array}$$

We want to find an as large as possible value for β^κ such that it bounds the optimal value for a feasible solution. This notation means that if $(x,\bar{y}^\kappa)$ is a feasible solution and x belongs to its domain D_x, this implies that the objective value of the subproblem is at least β^κ. We want to infer an as good as possible lower bound for the original problem such that if $y = \bar{y}^\kappa$, we have $f(x,y) = \beta^\kappa$. We therefore need to construct a function $\beta^\kappa_{\bar{y}}(y)$ such that if $y = \bar{y}^\kappa$, then $f(x,y) = \beta^\kappa$. This is done with the aid of a variable α in the objective function of the master problem. The master problem is defined as follows (in iteration ν):

$$\begin{array}{ll} \text{minimize} & \alpha \\ \text{subject to} & y \in D_y \\ & \alpha \geq \beta^{\kappa}_{\bar{y}}(y) \quad \kappa = 1, \ldots, \nu - 1 \end{array}$$

In this problem, $\alpha \geq \beta^{\kappa}_{\bar{y}}(y)$ is the Benders cut from iteration κ. The construction of $\beta^{\kappa}_{\bar{y}}(y)$ differs per context, but it should at least hold that $\beta^{\kappa}_{\bar{y}}(\bar{y}^{\kappa}) = \beta^{\kappa}$. Beside this, it should be a valid lower bound. We therefore must have $f(x, y) \geq \beta^{\kappa}_{\bar{y}}(y)$ for all feasible (x, y).

A possible way to define $\beta^{\kappa}_{\bar{y}}(y)$ is as the so called no-good cut. A no-good cut just cuts out the solution $\bar{y}^{\kappa}(y)$. For MILP with binary variables, this cut is defined as follows:

$$\alpha \geq \beta^{\kappa}\big(\sum_{i:\bar{y}^{\kappa}_i=1}(y_i - 1) - \sum_{i:\bar{y}^{\kappa}_i=0}(y_i) + 1\big)$$

In combination with $\alpha \geq 0$, this is a valid logic-based Benders cut. If $y = \bar{y}^{\kappa}$, we have $\alpha \geq \beta^{\kappa}$ and $\alpha \geq 0$ in all other cases. Such a no-good cut is in many cases not very helpful since it only limits one particular solution for the master problem. It is desirable that more solutions are cut by each Benders cut, but this needs problem specific construction. In the case of the yard crane scheduling problem we can limit the number of y variables in the Benders cut due to decomposition to different subproblems. This is explained in the next section.

6.3 Logic-based Benders for Model1

In the previous section we described the logic-based Benders decomposition method. In this section we implement this method for Model1. We split the problem between the x and the y variables. This leaves us with an assignment problem and the yard crane scheduling problem for one single crane within a zone. We construct a master problem which has a lot of similarities to an assignment problem with the y variables. Once these y variables are fixed we have m (the number of cranes) different subproblems that are independent of each other. This results in multiple subproblems, but with a smaller problem size. Since the runtime grows exponentially in the problem size, it seems advantageous to solve multiple small problems instead of one large problem. The objective is in the subproblems and not in the master problem. That means the algorithm just finds a feasible assignment for the y variables, not knowing what the delay of the subproblems will be. Therefore the master problem might introduce poor assignments in early stages of the algorithm. The master problem gets only feedback on performance after solving the subproblems in an iteration. In an early stage the master problem might suggest unrealistic assignments (many jobs to one crane, few to another), resulting not only in bad optimal values for the subproblems, but also in a long runtime for the large subproblems. Later in this section we will introduce a guiding objective for the master problem and some lower bound procedures that should prevent (in many cases, not all)

this from happening. This decomposition method is based on the logic-based Benders cuts (Hooker and Ottosson (2003)) similar to Sarmad et al. (2013).

6.3.1 Problem definitions

The problem is decomposed into a master problem similar to an assignment problem and subproblems that represent special cases of a traveling salesman problem with time windows. In every iteration, and for every subproblem, a logic-based Benders cut is introduced to the master problem. These logic-based Benders cuts represent the optimal values of the subproblems already discovered. The cuts teach the master problem that assigning a specific set of jobs to a crane will result in at least λ_k delay (where λ_k is the optimal value of subproblem k in iteration κ). Define the set $\mathcal{C}_k^\kappa = \{i : y_{ik}^\kappa = 1\}$ including all variables y_{ik} which were equal to 1 in iteration κ. The master problem is defined as follows:

$$\text{Minimize} \quad \sum_{k=1}^{m} \alpha_k$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1, ..., n$$

$$\sum_{l=1}^{m} y_{jl} \cdot l - \sum_{k=1}^{m} y_{ik} \cdot k \geq f(b_i, b_j) \qquad i, j = 1, ..., n; i \neq j$$

$$\alpha_k \geq \lambda_k^\kappa \big(\sum_{i \in \mathcal{C}_k^\kappa} y_{ik} - |\mathcal{C}_k^\kappa| + 1 \big) \qquad k = 1, ..., m; \kappa = 1, ..., \nu - 1$$

$$y_{ik} \in \{0, 1\} \qquad i = 1, ..., n; k = 1, ..., m$$

Note that as in Model1 we have:

$$f(b_i, b_j) = \begin{cases} 0 & b_j \geq b_i - SEP; \\ -m & \text{otherwise.} \end{cases} \qquad i, j = 1, ..., n$$

In this formulation, λ_k^κ is the optimal value of a subproblem solved in iteration κ. The Benders cut is defined in those variables y_{ik} that were equal to 1 in a specific iteration for a certain k. Here we exploit the multiple subproblems. There are k different Benders cuts introduced per iteration. Those Benders cuts involve only a few variables y_{ik} which makes them exclude many feasible solutions. These Benders cuts define a penalty coming from assigning a certain set of jobs to a certain crane. A different implementation possibility is to add more cuts also for assigning the same set of jobs to other cranes. That will lead to k^2 constraints per iteration. In some cases this reduces the computation times, but in other cases it does not.

The subproblems are defined for a single crane and are equal to the formulation of Ng (2005). Other structures (more similar to a TSP) could be introduced, but we stick to

this one because it seems to perform well in experiments. Subproblem$_k$ is defined as follows:

$$\text{Minimize} \sum_{i=1}^{n_k} t_i - (r_i + h_i) \tag{6.3}$$

$$x_{ij} + x_{ji} = 1 \qquad i = 1, ..., n_k - 1; j = i + 1, ..., n_k \tag{6.4}$$

$$t_j - t_i - Mx_{ij} \geq d_{ij} + h_j - M \qquad i, j = 1, ..., n_k;\ i \neq j \tag{6.5}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, ..., n_k \tag{6.6}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, ..., n_k; i \neq j \tag{6.7}$$

Where M is defined as in Model1 ($M = \max_{i=1..n}(r) + n(\max_{i,j=1..n}(d) + \max_{i=1..n}(h))$). Note that the problem size of the subproblem depends on the number of jobs assigned to a crane by the master problem (n_k). Only the variables x corresponding to two jobs that are assigned to a specific crane are present in the subproblem for this crane. This significantly reduces the number of binary variables x in the subproblems. Note that we really need the delay as an objective value instead of just the sum of completion times as could be used by Model1, Model2 and Model3. We want to direct the master problem to choose subproblems with small delay which do not necessarily correspond to problems with a small completion time. In this decomposition the master problem finds a lower bound for the original problem in every iteration. The sum of the objective values of the subproblems provide an upper bound for the original problem in all iterations. Note that the lower bounds produced in subsequent iterations form a monotonically increasing sequence. The upper bounds found do not have such a structure. The upper bounds found in subsequent iterations might increase or decrease. If the upper bound is equal to the lower bound in a certain iteration, an optimal solution is found.

Algorithm 1 Logic-based Benders decomposition

```
while LB<UB do
    Solve MP
    Update LB
    Create SPs given y
    Solve SPs
    Update UB
    Add lb-cuts to MP
end while
```

6.3.2 Adjustments

Algorithm 1 performs poorly because some subproblems take longer to solve than the entire problem at once. This happens when unrealistic subproblems are produced, for example a subproblem where a yard crane has to perform 30 jobs in one hour instead of around 10 jobs per hour. If one yard crane has to perform too many jobs, it will result

in much delay. As noticed before, the runtime of Gurobi for a problem is very dependent on the amount of delay that is found. This might be due to the larger LP-relaxation gap. Gurobi solves an instance with 3 cranes and 30 jobs faster in general than an instance with 1 crane and 25 jobs. We will make two adjustments to the algorithm to try to prevent this from happening. First we will introduce a small extra objective in the master problem that adds a value between zero and one. Rounding down the objective value afterwards will result in the same optimal solution. The master problem finds the best assignment currently possible during the algorithm, but when several assignments lead to the same objective value it is directed to choose the one that distributes the jobs among the cranes "fairest". We add a continuous variable β to the problem. The following constraints for β are added: $\beta \geq \sum_{i=1}^{n} y_{ik}$, for $k = 1, \ldots, m$. In this way, β corresponds to the maximum number of jobs assigned to a crane when minimizing over β. In the objective, β is scaled down by $n + 1$ to guarantee not to interfere with logic-based Benders cuts in finding an optimal solution. So we add the term $\frac{\beta}{n+1}$ to the objective function. During the minimization, the part $\sum_{k=1}^{m} \alpha_k$ is most important because this minimizes over integral values. Decreasing $\sum_{k=1}^{m} \alpha_k$ with 1 will be preferred above reducing $\frac{\beta}{n+1}$. Only when several solution lead to the same value for $\sum_{k=1}^{m} \alpha_k$, the solution with the smallest value for $\frac{\beta}{n+1}$ will be chosen.

Although the probability that an unrealistic assignment is calculated by the master problem is reduced by adding this term to the objective function of the problem, it is certainly not prevented. We will illustrate this by an example. Assume we have a problem with three cranes, 30 jobs and the jobs are numbered from left to right. Assume that in the first iteration the master problem assigns 10 jobs to each of the cranes $1, 2$ and 3. The ready times and handling times of jobs are absent in the master problem, it is only concerned with the delay that assigning certain jobs produces. Assume that the subproblems find a delay of 100, 13 and 124 for this assignment. During the next iteration the master problem assigns the leftmost 9 jobs to crane 1 and the rightmost 9 jobs to crane 3. The other 12 jobs are assigned to crane 2 and therefore the master problem aims to score a total delay of 13. Crane 2 will lead to a delay of at least 13, but crane 1 and 3 might find 0 delay as a result of having to perform one job less. Assume that in the second iteration the subproblems find a delay of 60, 34 and 124 for this new assignment. Again, more jobs are added to the second crane and this can be repeated several times. In the master problem we do not know the ready times of jobs. While we find a small delay for a large set of jobs for crane 2, adding two new jobs to the zone of crane 2 might result in much delay. When the crane falls behind of schedule, delay might add up. This results in a very difficult and large subproblem.

The second adjustment tries to prevent the algorithm to solve difficult subproblems. Assuming that a difficult subproblem will result in much delay, we try to find a lower bound first. We will refer to this lower bound as sublb to avoid confusion with the lower bound for the objective of the entire problem found by the master problem. If a sublb is larger than an upper bound found in earlier iterations, this crane assignment will never result in an optimal solution. In addition, for the master problem it is restrictive enough

to add a cut using this sublb instead of the optimal value. With the aid of this sublb, the master problem knows that assigning this set of jobs to the concerned crane will not lead to a better solution. Using sublb in this manner will not destroy the optimality guarantee. To find a sublb we propose several procedures that use relaxations of the single crane problem. Unfortunately we cannot use an LP-relaxation because this will result in zero delay. This will not lead to a sublb that is larger than the best upper bound found by the algorithm. For a good procedure for finding a sublb it is desired that it has a very small runtime and finds a bound as close as possible to the optimal value. Below we will introduce three procedures that find a sublb. We will refer to these procedures as sublb procedures.

6.3.3 Space relaxation heuristic

In this first sublb procedure we use a space relaxation. We discard the spatial parameter in our model. We also set the handling times of the jobs equal to the minimum handling time. When all jobs are at the same bay and have the same handling time, a "greedy algorithm" will find an optimal solution efficiently. We sort the jobs on earliest possible finish time and handle the jobs in this order. The crane will not be idle when jobs are available and when delay occurs it minimizes the number of jobs that are delayed at that moment by finishing a job as soon as possible. This results in the following procedure:

Algorithm 2 Sublb procedure 1

$h = \min_i h_i$
$f_i = r_i + h, \forall i$
Sort jobs on f_i
$t_1 = f1$
for $j = 1 : n_k - 1$ **do**
    $t_{j+1} = \max f_{j+1}, t_j + h$
end for
Delay $= \sum_{i=1}^{n_k} (t_i - (r_i + h_i))^+$

In some cases, this procedure finds a bound close to the optimal value. In other cases (especially when jobs are far apart in space) it finds very bad bounds. The advantage of this procedure is that it is guaranteed to have a very small runtime. Since sorting can be done in $\mathcal{O}(n \log n)$ and other functions can be done in $\mathcal{O}(n)$ this procedure finds solutions in $\mathcal{O}(n \log n)$ time.

6.3.4 Time relaxations heuristic

In this procedure a sublb is found by halving the problem size. We introduce an artificial second crane for the problem and split the problem into two new problems. Therefore we sort the jobs according to increasing ready time and assign the first $a = \text{floor}(\frac{n}{2})$ jobs to the first problem and the other $n - a$ jobs to the second problem. While this reduces the problem size by a factor of two, two problems have to be solved. Since the worst-case runtime increases exponentially in n, solving two smaller problems results in a smaller worst-case runtime. Experiments show that indeed the computation time of solving the smaller problems is shorter than the computation time of solving the large problem. Adding the delays of both new problems gives a valid sublb. If the delay adds up in the large problem around job a, the two small problems will find less delay than the large problem. The new subproblems together will never find more delay than the original subproblem. Let t'_i denote the completion time of job i in the small problems. Let λ_k denote the delay found by the larger problem and λ_k^1 and λ_k^2 denote the delay found by the smaller problems. We have the following inequality since the t'_i will result from optimal schedules for those jobs:

$$\lambda_k = \sum_{i=1}^{n} t_i - (r_i + h_i) \tag{6.8}$$

$$= \sum_{i=1}^{a} t_i - (r_i + h_i) + \sum_{i=a+1}^{n} t_i - (r_i + h_i) \tag{6.9}$$

$$\geq \sum_{i=1}^{a} t'_i - (r_i + h_i) \sum_{i=a+1}^{n} t'_i - (r_i + h_i) \tag{6.10}$$

$$= \lambda_k^1 + \lambda_k^2 \tag{6.11}$$

Algorithm 3 Sublb procedure 2

Sort jobs on increasing r_i
Set $a = \text{floor}(\frac{n}{2})$
Solve for first a jobs
Solve for last $n - a$ jobs
Add delay found

This procedure finds a strong sublb when delay is not added up around job a in the optimal solution. The runtime is significantly shorter than solving the subproblem, but it is not efficient like procedure 1. This means the runtime is not polynomial bounded.

To solve the problem when delay is adding up around job a, we introduce a third procedure very similar to the second one. Instead of splitting the jobs into two sets, we split them into three. The rest of the procedure is the same. It might find a stronger sublb when delay especially occurs around job a.

6.3.5 Algorithm

The algorithm presented next will solve a problem instance using all three procedures (if necessary) to avoid solving large subproblems. For subproblems larger than $n = 15$ it will first solve sublb procedure 1. When a valid sublb is found ($sublb >$ UB) this value is used for the logic-based Benders cut. If the sublb is not valid, the second sublb procedure is tried. If this again fails to find a valid sublb, procedure 3 is invoked. If none of the sublb procedures finds a valid sublb, the subproblem is solved to optimality by Gurobi. This idea is given in pseudo-code for algorithm 4.

Performance might be increased when better sublb procedures are available. It is not uncommon that all three procedures will find a sublb of a factor 2 below the real optimal value of he subproblem. It might be advantageous if lower bounds can be derived while solving the subproblems. If an algorithm can guarantee lower bounds while searching for an optimal value of the single crane subproblem these can be used if they are larger than the best found upper bound. The algorithm can terminate its search for an optimal value since this lower bound suffices.

Algorithm 4 Logic-based Benders decomposition

```
while floor(LB)<UB do
    Solve MP
    Update LB
    for k = 1 : m do
        Create SP_k given y
        if n_k > 15 then
            Solve sublb procedure 1
            if sublb1>UB then
                Create cut using sublb1
            else
                Solve sublb procedure 2
                if sublb2>UB then
                    Create cut using sublb2
                else
                    Solve sublb procedure 3
                    if sublb3>UB then
                        Create cut using sublb2
                    else
                        Solve SP_k
                        Create cut using SP_k
                    end if
                end if
            end if
        else
            Solve SP_k
            Create cut using SP_k
        end if
        Add cut to MP
    end for
    if then ∑_{k=1}^{m} D_k < UB
        Update UB
    end if
end while
```

Chapter 7

Rolling horizon

In the previous sections, we studied the problem for a short fixed planning window. Business on the terminal is ongoing and different planning windows need to be well connected. As we have seen, the runtime of the algorithms is very dependent on the length of the planning window (and as a result the number of jobs). Those two observations suggest the use of rolling horizon algorithms. In rolling horizon algorithms, decisions are made based on knowledge of the near future. Using knowledge of the future comes at a certain cost or information is unreliable. The further ahead one takes information into account, the higher the costs or the information becomes less reliable. The idea behind a rolling horizon is to look a certain number of time periods ahead while making decisions. Those time periods are included in the planning window. Then in the next time period, another time period is added to the information and new decisions can be made. The boundary on information to incorporate rolls ahead while advancing in time. When a long planning window is split into different time periods there arise problems on the boundary between the different time periods. By saving only part of the solution and search for a new solution again, this problem is reduced.

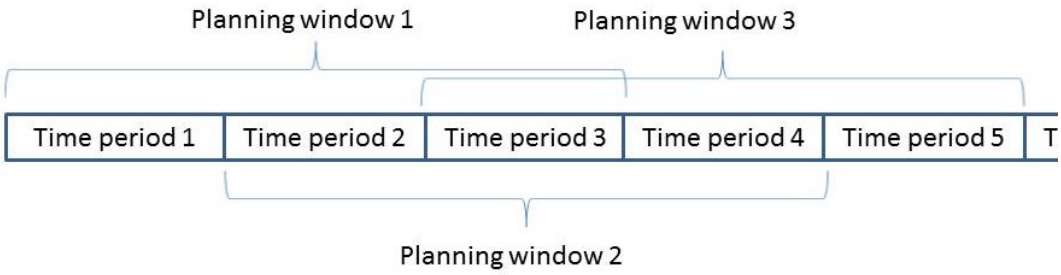


Figure 7.1: Time periods that are included in the planning window

From the past we only need the end position of the cranes, completion time of their

latest job and the jobs that where not handled within the time period. Jobs that are delayed beyond the boundary of the time period are postponed to the next time period. Otherwise they have to be within the zone of a crane while this might not be desirable for the next time interval. It might be advantageous to let another crane perform the job.

In the first section we will give a more formal formulation of the framework. After this we introduce two different algorithms within this framework. The first algorithm uses the expected arrival times of the jobs as deterministic input for the coming 2 or more time periods. It creates a schedule for the coming 2 (or more) time periods where the schedule is only performed for the current time period. The advantage is that zones are fixed such that they are better positioned for the next time interval. However, fixing the intervals for a longer period might be worse for the first time interval. It also increases the runtime of the algorithm because the problem that needs to be solved is much larger.

The second algorithm solves the current and the next time interval iteratively. The current time interval is solved and then the next time interval given the schedule of the current time interval is solved. The additional delay coming from the start position of the cranes in the next time interval is used as a penalty in the problem for the current time interval. This is a decomposition for solving the upcoming two time intervals. This method can be extended to incorporate more time intervals. Solving the second time interval is then again done by calling the decomposition algorithm on the second and the third time interval. This drastically increases the computation times due to nested loops, but incorporates more information about the future. The method is explained in detail in Section 7.2.

For solving individual problems we use the static version described before. We can either choose to use the Benders decomposition method or solve an individual problem with Model1 directly. A third option is to store solutions found by the algorithm. A schedule is made for a longer period than is actually executed. The part of the schedule that is not executed can be used as a starting point in the next iteration. There is a fair chance that this is still a good solution, so using it as a starting point for the branch and bound algorithm of Gurobi might improve performance.

7.1 Rolling horizon algorithm

For the rolling horizon algorithm we use two time parameters. $Tsave$ is the length of a time period. After a full iteration of the algorithm, a schedule is made for the coming $Tsave$ seconds. The other parameter is $Twindow$. In an iteration, the algorithm solves for the coming $Twindow$ seconds. This means that all jobs with $r_i < Twindow$ are included in the planning window. After finding a schedule, the algorithm saves the schedule for the coming $Tsave$ seconds and a new schedule is made from this starting

point. In the view of Figure 7.1 this means that $Tsave$ corresonds with a time period and $Twindow$ with a planning window. Note that $Twindow$ does not have to be a multiple of $Tsave$ as Figure 7.1 suggests. This however is necessary for the time decomposition algorithms as is presented in the next section.

After an iteration, the ready times of the jobs and finish times of the cranes are shifted back with $Tsave$, such that new jobs fall into the interval $[-\infty, Twindow]$. Jobs that could not be scheduled in time in the previous time interval have negative ready times, but should be included again in the new problem. As a consequence the number of jobs in an iteration might increase during to stacked delay. This might result in long computation times for certain iterations. This is a downside, but if we save the schedule for all jobs originally in $[0, Tsave]$ we have some cranes with a schedule in the next iterations which might interfere with the schedules in that time period. Besides this, it might be advantageous to perform a job at the beginning of the second time interval before a job at the end of the first time interval. $Twindow$ can be chosen large such that many jobs are included into the problem and the algorithm can adjust the schedule well when workload is not uniformly divided over the bays. This foresight helps with moving the cranes in time to the proper location. A downside is that the algorithm fixes the zones for a long time even though they are updated again after $Tsave$. The zone fixing might not give the best solution for the $[0, Tsave]$ time interval. If $Twindow$ is chosen small (equal to $Tsave$), then the problems to solve remain small, are solved fast and zones are not fixed for long periods. On the other hand, the algorithm has no knowledge about expected workload in the future. If there are many jobs in the first 10 bays, most cranes will be moved to this area. But if after a few minutes workload shifts to the last 10 bays, the cranes are not positioned in time. The following pseudo-code illustrates how the algorithm works.

Algorithm 5 Rolling Horizon

```
for T=1:ceil(planningwindow/Tsave) do
    for i=1:n do
        if r_i < Twindow then
            add job i to problem
        end if
    end for
    Solve problem
    if job t_i is scheduled and t_i < Tsave then
        save t_i
    end if
end for
```

7.1.1 MILP

First we need to introduce some new constraints to deal with the start positions of the cranes. From the previous problem, we have a location and finish time of each crane. The schedule for the current time period has to take those times and positions into account. An easy way to model this is to put the last jobs of the previous schedule in the new problem and fix their finish times and crane assignment. This would however limit the zone fixing possibilities for the current time periods. We therefore introduce two different constraints that are concerned with the start location and the time that these cranes become available. We will refer to the start location by b_{0k} and the time the crane is available by t_{0k} for every crane k.

The first additional constraint limits cranes to start with a job before they can reach them from the current location. If job i is at another bay than b_{0k}, then crane k needs at least $|b_{0k} - b_i|$ time units to move and cannot start job i earlier than $t_{0k} + |b_{0k} - b_i|$. The finish time of job i is therefore limited to:

$$t_i \geq (h_i + t_{0k} + |b_i - b_{0k}|)y_{ik}, \text{ for } i = 1, \ldots, n; k = 1, \ldots, m$$

Note that h_i, t_{0k}, b_i and b_{0k} are all fixed and known before solving the problem. Therefore the above constraint is still a linear constraint. The next constraint deals with the non-crossing of the cranes due to the start positions. If t_{0k} is very small, $t_{0k} + |b_{0k} - b_i|$ can be smaller than zero even though the location of job i is not very near the start position of crane k. For crane k it might be possible to start with job i at the beginning of the current time interval, because it can use its idle time at the end of the previous time interval to move to location b_i. However, it might be impossible due to the positioning of another crane. If crane $k+1$ is still working on a job at the end of the previous time interval between b_{0k} and b_i, for crane k is not possible to move during the previous time interval. This is modeled with the following constraint.

$$\sum_{i \in B_k^+} \sum_{l=1}^{k-1} y_{il} + \sum_{i \in B_k^-} \sum_{l=k+1}^{m} y_{il} = 0, \text{ for } k = 1, \ldots, m$$

$$B_k^+ := \{i : i \in \{1, ..., n\}, b_i + \frac{1}{4}\min(0, t_{0k} - r_i) \geq b_{0k} - SEP\}$$

$$B_k^- := \{i : i \in \{1, ..., n\}, b_i - \frac{1}{4}\min(0, t_{0k} - r_i) \leq b_{0k} + SEP\}$$

In this constraint we use the sets B_k^+ and B_k^- for every k to give specific conditions. Remember that the constant SEP denotes the number of bays that need to be empty between two yard cranes. The set B_k^+ contains all b_i that are larger than the initial position of crane k minus the separation constant SEP. Note that crane k can move away $\frac{1}{4}$ such that the "blocked area" moves with yard crane k. So B_k^+ denotes the set of bays that are blocked by yard crane k for yard cranes at the left of yard crane k (yard cranes with a number smaller than k). This set is defined on the initial time of yard

crane k and the ready times of the jobs. There are no time variables in the set, so the constraint stays a linear constraint. The set B_k^- is defined similarly and denotes the set of bays that is blocked by crane k for cranes with a higher crane number. Only the fixed ready time of job i is used. This is a downside because there are situations possible that it is advantageous to let crane $k-1$ perform job i although it has to wait until crane k has moved away to an even higher bay number. But using the ready time keeps the constraint linear and already generates much more flexibility than fixing the zones around the last job of the previous time interval. This leads to the following MILP that needs to be solved for every time interval. For definition of the variables and constants, see Section 5.2.

$$\text{Minimize:} \quad \sum_{i=1}^{n} \big(t_i - (r_i + h_i)\big) \tag{7.1}$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1, 2, ..., n \tag{7.2}$$

$$\sum_{l=1}^{m} y_{jl} \cdot l - \sum_{k=1}^{m} y_{ik} \cdot k \geq f(b_i, b_j) \qquad i, j = 1, ..., n; i \neq j \tag{7.3}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, ..., n \tag{7.4}$$

$$y_{ik} + y_{jk} - 1 \leq x_{ij} + x_{ji} \qquad i, j = 1, ..., n; i \neq j; k = 1, ..., m \tag{7.5}$$

$$t_j - t_i \geq d_{ij} + h_j - (1 - x_{ij})M \qquad i, j = 1, ..., n; i \neq j \tag{7.6}$$

$$t_i \geq (h_i + |b_i - b_{0k}| + t_{0k})y_{ik} \qquad i = 1, ..., n; k = 1, ..., m \tag{7.7}$$

$$\sum_{i \in B_k^+} \sum_{l=1}^{k-1} y_{il} + \sum_{i \in B_k^-} \sum_{l=k+1}^{m} y_{il} = 0 \qquad k = 1, ..., m \tag{7.8}$$

$$x_{ij}, y_{ik} \in \{0, 1\} \qquad i, j = 1, ..., n; i \neq j; k = 1, ..., m \tag{7.9}$$

7.1.2 Delay

It seems natural to expect that less overall delay is found, when $Twindow$ is large. A large planning window makes sure the algorithm can adjust well to changes in the job distribution among the bays through time. However, it is not always true that a larger $Twindow$ leads to less delay. Sometimes fixing the zones for this longer period creates worse solutions for the first part $[0, Tsave]$. Zones for jobs after $Tsave$ are determined again in the next iteration, but during the current iteration the algorithm does not know that this is possible. This issue can be reduced by giving different weights based on the ready times of jobs. Jobs that are in the near future can be given higher weights than jobs with a large ready time.

It is also interesting to note that the algorithm used to solve the problem in an iteration can generate different overall solutions. For instance, the Benders decomposition algorithm and direct solving both find an optimal solution for Model1. However, they might find a different solution with the same optimal value. This can lead in a next iteration to different start position of the cranes and therefore different overall delay found by the algorithms.

7.2 Time decomposition

A second way to divide the planning window into multiple time periods is via a time decomposition. This idea has some similarities to the Benders decomposition. It can be implemented in a rolling horizon fashion. First, the current time period is solved (for instance the next 5 minutes). Then, the next time period is solved (for instance from 5 to 10 minutes) given the solution of the current time period. The end positions of the cranes in the current time period influence the result of the next time period. The difference in delay for the next time period given the current solution of the current time period and the delay of the next time period without start position constraints is used for a cut for the current time period. This cut "teaches" the problem of the current time period that choosing this crane to job assignment results in a certain amount of extra delay. More feasible solutions are tried for the current time period to try to minimize the total delay over the current and the next time period. As cut we have to use a so called "no good" cut including all binary variables of the current time period to ensure optimality. It might be advantageous for the next time period to switch a job from one crane to another or to switch two jobs within one yard crane zone.

Because such a "no good" cut will result in many iterations we use a cut only based on the y variables determining the job to crane assignment. This reduces the possibilities for the current time period and therefore the number of iteration needed in the decomposition algorithm. In this way optimality for the two investigated time periods is not guaranteed, but much less delay is found compared to solving the coming two periods together at once. In 6, the pseudo-code of the algorithm used for the Time Decomposition is given.

7.2.1 Cut-generation

The time-cut is generated in each iteration of the while loop. As described above, we only cut for the y-variables to avoid too many iterations. The cut will put a penalty on the currently found job to crane assignment. This penalty (P) equals the difference between the objective value of solving the second problem with start constraints and without start constraints. This keeps the penalty low, but valid. If we did not subtract the delay without start constraints, the number of iterations would increase. The penalty is only available for job to crane assignments that are tried before. If the penalty is large, the algorithm will try many suboptimal solutions for the first problem to minimize the delay over the first and second problem combined. This is unnecessary because the second problem will always encounter at least the delay of the problem without start constraints. Let $\bar{y}_{ik}$ be the job to crane assignment for problem1 in the current iteration. Then we generate the following time-cut:

$$\sum_{i=1}^{n}\sum_{k=1}^{m} P \cdot \bar{y}_{ik} \cdot y_{ik} - \alpha \leq (n-1) \cdot P \tag{7.10}$$

Algorithm 6 Time decomposition

```
for T=1:ceil(planningwindow/Tsave) do
    for i=1:n do
        if r_i < Tsave then
            add job i to problem1
        end if
    end for
    for i=1:n do
        if Tsave < r_i ≤ 2 · Tsave then
            add job i to problem2
        end if
    end for
    Solve problem2 without start constraints
    MinObj2=Obj2
    while LB < UB do
        Solve problem 1
        LB = Obj1 + MinObj2
        Calculate start postions cranes problem 2
        Solve problem 2
        UB = Obj1 − α + Obj2
        Create time-cut based on P = Obj2 − MinObj2
        Add time-cut to problem1
    end while
    if job t_i is scheduled and t_i < Tsave then
        save t_i
    end if
end for
```

If $\bar{y} = y$ in the next iteration, the left-hand side becomes $n \cdot P$ and the right-hand side $(n-1) \cdot P$. Therefore α needs to be at least P to ensure feasibility. The variable α is introduced into the objective with weight 1.

7.2.2 Multiple decomposition

It is also possible to do a decomposition twice or even more times. This idea is worked out in pseudo-code in Algorithm 8. During an iteration the second time period is solved. First as a benchmark and later to determine the extra delay coming from the schedule for the first time period. Instead of just solving the second time period (ignoring the future beyond $2 \cdot Tsave$), we can solve the second time period by again calling the time decomposition algorithm. By doing this iteratively we can increase the depth of the time decomposition algorithm. We need to give the algorithm a maximal depth (Mdepth) such that if it reaches this depth it stops calling for the decomposition algorithm and

just solves the last time interval on its own. The algorithms consists of an outer loop (7) with the rolling horizon approach, calling on an algorithm (8) including the time decomposition until a certain depth.

Algorithm 7 Rolling Time Decomposition

```
for T=1:ceil(planningwindow/Tsave) do
    Do TimeDecomposition{depth = 0} return t, y
    for i=1:n do
        if t_i ≤ Tsave then
            Save t_i, y_ik
        end if
    end for
    Calculate start positions cranes for next interval
end for
Calculate objective using t, y
```

Note that the depth variable is important for the tasks that the sub-algorithm TimeDecomposition needs to perform. The jobs that come into a specific time period are based on their ready times. The jobs under consideration in the time period are the jobs with $r_i \in [depth \cdot Tsave, (depth+1) \cdot Tsave]$. The exception is for depth equal to zero, because in this case all jobs with $r_i \in [-\infty, Tsave]$ need to be in the time period (also those with negative ready times). If the depth equals Mdepth, the problem needs to be solved directly without calling for the TimeDecomposition function again. The solution that is saved is again all jobs completed before $Tsave$. So, jobs scheduled in the first time period that are finished after $Tsave$ are scheduled again in the next time interval. This is to guarantee that cranes do not cross each other. In suggested solution for more time intervals this is not the case. All jobs are scheduled and the completion time is taken to the next time interval. This might result in wrong solutions and therefore less or more delay. However, for the final schedule this is avoided.

In the above description, the algorithm assigns the same value to the time periods under consideration. It can be advantageous to give a higher weight to jobs arriving in the first time period and decrease this weight factor along with further time periods. In the end, the algorithm saves only the solution for the coming time period. It is most important that few delay is found for this time period because the other time periods will be including in the next iteration of the rolling horizon. Schedules for later time periods might very well be adjusted later. This weight can be implemented with use of the bounds found by the algorithm. For example, we can multiply $MinObj2$, $Obj2$ and P by $w = \frac{1}{3}$ in the above algorithm.

Algorithm 8 Multiple time decomposition

if depth=0 **then**
  for i=1:n **do**
    if $r_i < Tsave$ **then**
      add job i to problem
    end if
  end for
else
  for i=1:n **do**
    if $depth \cdot Tsave \leq r_i < (depth + 1) \cdot Tsave$ **then**
      add job i to problem
    end if
  end for
end if
if $depth = Mdepth$ **then**
  Solve Problem **return** $t, y, Obj1$
else
  Do TimeDecomposition$\{depth = depth + 1, t_0, b_0\}$ (without start constraints) **return** $t, y, Obj2$
  $MinObj2 \leftarrow Obj2$
  while $LB < UB$ **do**
    Solve problem **return** $t, Y, Obj1$
    $LB = Obj1 + MinObj2$
    Calculate t'_0 and b'_0 (start positions cranes next problem)
    Do TimeDecomposition$\{depth = depth + 1, t'_0, b'_0\}$ **return** $t, y, Obj2$
    $UB = Obj1 - \alpha + Obj2$
    Create time-cut based on $P = Obj2 - MinObj2$
    Add time-cut to problem
  end while
  $Ojb1 \leftarrow LB$
end if
return $t, y, Obj1$

Chapter 8

Stochastic input

In the previous sections all input was assumed deterministic, meaning that we assumed that everything was known beforehand without any uncertainty. The introduction of the rolling horizon algorithms create a good framework for a stochastic model. We can incorporate uncertainty about the job arrival times in the model without creating many complications for practical use. It is undesirable that a yard crane goes to a bay expecting a job that will not be there. With the aid of the rolling horizon principle, we can differentiate between jobs in the coming minutes and jobs further ahead in time. We will use this for introducing stochastic arrival times of jobs. In practice, it can be difficult to determine exact arrival times of jobs. Especially when they are dependent on external factors it is hard to predict their actual arrival. We will not take uncertainty into account due to terminal operations (for example, a traffic congestion of internal trucks). Uncertainty coming from outside the terminal has a less direct influence on operations and is more manageable to include. We will introduce uncertainty for the arrival of trucks at the gate. Trucks have a certain planned arrival time, but often they are delayed due to traffic jams or other external factors. Seaside operations are less sensitive to unexpected lateness. Communication between the vessel and the terminal is easier to manage compared to communication between trucks and terminal. The loading and unloading activities are controlled by terminal operations and therefore the arrival times of jobs are easier to predict.

Since the structure of the time decomposition algorithm is useful for modeling uncertain variables, we use this in our stochastic model. The structure makes it possible to plan for a short time interval where the actual ready times of jobs are reasonably certain, and use uncertain ready times in the other time intervals. This way schedules can be formulated that anticipate well on the uncertainty.

8.1 Stochastic model

To model the uncertainty of the job ready times we use the planned arrival time of the jobs with a lateness (not to be confused with delay of jobs due to scheduling of yard cranes) that is distributed as an exponential distribution with parameter λ to be specified later. Because the largest uncertainty comes from outside the container terminal, we assume that the job ready time is deterministic once they are within the boundaries of the container terminal. For receivel and delivery moves this means that trucks have a specified planned arrival time at the gate. Their arrival at the gate can be later than planned but once at the gate, their arrival time at the yard will be quite certain. Let $\bar{a}_i$ denote the planned arrival time of job i at the gate and $a_i = \bar{a}_i + l_i$ be the actual arrival time of job i at the gate. The value l_i denotes the time that the truck arrives too late due to external factors. The quantity l_i is uncertain and we assume it comes from a probability distribution. We will first introduce a probability space to define the random variable L_i. Let $(\Omega, \mathcal{F}, P)$ denote a probability space where Ω is the sample space, $\mathcal{F}$ a sigma algebra and P a probability measure. It is natural to assume that the truck arrival times at the gate are real valued. The observation space is therefore the real numbers ($\mathbb{R}$) equipped with the Borel σ-algebra $\mathcal{B}(\mathbb{R})$. Let $L_i : (\Omega, \mathcal{F}) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ for $i = 1, \ldots, n$ be i.i.d. random variable and denote by $l_i \in \mathbb{R}$ their outcome. We denote with $P_{L_i}(A) = P(\omega : L_i(\omega) \in A)$ the probability of the event $\{\omega : L_i(\omega) \in A\}$ and let $F_{L_i}(t) = P(L_i \leq t)$. Once the event $L_i(\omega) = l_i$ has taken place, $a_i = \bar{a}_i + l_i$ becomes known. Otherwise we have $a_i = \bar{a}_i + L_i$ and treat a_i as an uncertain quantity.

For scheduling it is important to identify which ready times are deterministic and which are uncertain. First we partition the coming planning window in different time periods. Let $I_j = [T_{j-1}, T_j]$ for $j = 1, \ldots, N$ denote N non-overlapping time periods, such that $\cup_{j=1}^{N} L_j = [0, T_{end}]$. Let A_{time} denote the time it takes algorithm A to find a schedule for the coming time period. Assume we are at time $T_{j-1} - A_{time}$ so we need to find a schedule for time period j. Jobs that are already past the gate are deterministic and their ready time is given by: $r_i = a_i + \tau_i$ where τ_i denotes the time it takes for a truck to reach its yard location from the gate. If job i is not completed yet, its ready time r_i will be in problem j. If at time $T_{j-1} - A_{time}$ we have not yet observed a_i (so job i has not arrived at the gate yet), we have an uncertain ready time for job i. We have the following definition for the arrival time at the gate at time T:

$$a_i = \begin{cases} \bar{a}_i + l_i & \text{if } \bar{a}_i + l_i < T \\ \bar{a}_i + L_i(\omega) & \text{if } \bar{a}_i > T \\ \bar{a}_i + \{L_i(\omega) | L_i(\omega) > T - \bar{a}_i\} & \text{else} \end{cases}$$

The time it takes for a truck to get from the gate to the yard might depend on the position of job i at the yard. The distance between gate and yard block can be different of each yard block. For simplicity we will assume constant travel time between yard and gate and define $\tau = \tau_i = 300s \ \forall i$. We define $\bar{r}_i = \bar{a}_i + \tau$ and $r_i = a_i + \tau$ as the planned job ready time and the actual job ready time. Since the latter is dependent on ω, we

assume an exponential distribution for L_i. We use the expected value of r_i at time T for scheduling purposes. If we have observed that $\bar{a}_i + L_i < T$, we have:

$$\mathrm{E}[r_i|\bar{a}_i + L_i < T] = \bar{a}_i + l_i$$

If on the other hand the truck has not arrived at the gate at time T we have:

$$\mathrm{E}[r_i|\bar{a}_i + L_i > T] = \mathrm{E}[\bar{a}_i + L_i + \tau|\bar{a}_i + L_i > T] = \bar{a}_i + \tau + \mathrm{E}[L_i|L_i > T - \bar{a}_i] = T + \tau + \mathrm{E}[L_i]$$

due to the memoryless property of the exponential distribution.

$$\begin{aligned}
\mathrm{E}[L_i|L_i > T - \bar{a}_i] &= \int_0^\infty l_i f_{L_i}(l_i|L_i > T - \bar{a}_i)\mathrm{d}l_i \\
&= \int_0^\infty l_i f_{L_i}(l_i - T + \bar{a}_i)\mathrm{d}l_i \\
&= \int_0^\infty (u + T - \bar{a}_i) f_{L_i}(u)\mathrm{d}u \\
&= T - \bar{a}_i + \mathrm{E}[L_i]
\end{aligned}$$

This leaves us with the following definition of r_i which we will use for our algorithms.

$$r_i = \begin{cases} \bar{r}_i + l_i & \text{if } a_i < T \\ \bar{r}_i + \mathrm{E}[L_i(\omega)] & \text{if } \bar{a}_i > T \\ T + \tau + \mathrm{E}[L_i(\omega)] & \text{else} \end{cases}$$

There are many different ways to take uncertainty into account in mathematical models. Some possibilities are using: distributions, simulation and oracles. When using distributions, we assume that the uncertainty is structured in a specific way. We can use knowledge about the expectation and variance to use in algorithms. Simulation is used when the expectation of a random variable is not a desirable input for the algorithm or when it is unknown (or highly dependent on different factors). Simulating different outcomes lead to different scenarios. The algorithm can find a solution under one of these scenarios. There are different choices that can be made with respect to the scenarios. One can choose the solution that finds the lowest delay as an average over all scenarios. Another choice would be to choose the solution that is optimal in most scenarios. In this way, the chance is largest that the chosen solution is indeed optimal for the actual realization. Oracles are used when there is a possibility to gain more information about uncertain values. Many times, this possibility comes at a certain price due to costs of further research or increased runtime of an algorithm. For our problem this oracle structure does not seem applicable.

8.2 Distributions

A straightforward way to deal with the uncertainty of ready times is to split between the certain ready times and the uncertain ready times. This can be done by choosing the

time periods such that time boundary ($Tsave$) of a planning period is between those two groups. We must at least have $Tsave \leq T$ to ensure that no uncertain jobs are saved to the final schedule. That might lead to yard cranes moving to bays while the job is not ready yet. A practical way to define time periods for scheduling is to use time periods of length $tr - A_{time}$. At time $T_{j-1} - A_{time}$ we have the following job ready times:

$$r_i = \begin{cases} a_i + tr & \text{if } a_i < T_{j-1} - A_{time} \\ \bar{a}_i + tr + \mathrm{E}[L_i(\omega)] & \text{if } \bar{a}_i > T_{j-1} - A_{time} \\ T_{j-1} - A_{time} + tr + \mathrm{E}[L_i(\omega)] & \text{else} \end{cases}$$

Note that if $a_i < T_{j-1} - A_{time}$, r_i is deterministic and its ready time is in the coming time period. On the other hand, if $a_i > T$ we have $P(r_i < T_j) = 0$ and therefore job j is not in the coming time period. This gives us the possibility to schedule for the coming time period as before in a deterministic way. If we incorporate time periods in the future, the ready times are uncertain. We can use the expectation of L_i to estimate delay of formed schedules for those time periods.

8.3 Simulating

Instead of using the expected ready time for later time periods, we can use simulation to find a schedule for the current time period. Even if we have a clear idea about the distribution of the ready times, simulation might be more advantageous than using individual expectation of the ready times. When many simulations can be used, the schedule based on the combination of expectations can be made. A schedule based on expectations can be very good if the ready times are close to their expectation but very bad if one particular job is very late. When we use simulations, the algorithm will detect that a certain solution has the risk to be very bad under certain realizations. A solution that is not optimal if jobs arrive according their expectation can be a better choice because it is more flexible to deal with some unwanted realizations. Simulation will make a more risk reduced choice.

We create N different realizations of the random variables in the uncertain time periods. In every realization an outcome is randomly generated from the exponential distribution for every job i. This can be implemented in the TimeDecomposition algorithm (see Algorithm 9). The current time period under consideration is again deterministic. The jobs that need to be handled are already beyond the gate, so their arrival time is assumed deterministic. The next time period consists of uncertain job arrival times. The same distributions as above are assumed but instead of scheduling on the expected value we create different scenarios. The algorithm will search for the best job to crane assignment based on the average of all different scenarios. So for a solution of the first time period, the extra delay coming from the start position of the cranes in the next time period is approximated by the mean under all scenarios. The best choice for the current time period is chosen based on this mean value.

Algorithm 9 Simulation

for T=1:ceil(planningwindow/Tsave) **do**
  for i=1:n **do**
    if $r_i < Tsave$ **then**
      add job i to problem1
    end if
  end for
  for j=1:N **do**
    clear problem2
    Simulate r^j
    for i=1:n **do**
      if $Tsave < r_i^j \leq 2 \cdot Tsave$ **then**
        add job i to problem2
      end if
    end for
    Solve problem2 without start constraints
  end for
  MinObj2=min(Obj2)
  while $\alpha \neq mean(Obj2) - MinObj2$ **do**
    Solve problem 1
    $LB = Obj1 + MinObj2$
    Calculate start postions cranes problem 2
    for j=1:N **do**
      clear problem2
      for i=1:n **do**
        if $Tsave < r_i^j \leq 2 \cdot Tsave$ **then**
          add job i to problem2
        end if
      end for
      Solve problem 2
      $UB = Obj1 - \alpha + Obj2$
    end for
    Create time-cut based on $mean(Obj2) - MinObj2$
    Add time-cut to problem1
  end while
  if job t_i is scheduled and $t_i < Tsave$ **then**
    save t_i
  end if
end for

In the time decomposition algorithm we can find the delay for the first two time intervals under every scenario. We can choose the schedule for the first time interval that lead to the smallest average delay over both time intervals. After a schedule for the first time

interval is found, the true realizations become known for the next time interval. Jobs that still have not arrived at the gate will not arrive this time interval and are scheduled in the next time interval (again under uncertainty).

Chapter 9

Results

In this chapter we discuss results of the testing of the algorithms for the different models. First we will discuss results for the static case (without the rolling horizon). After that, we discuss the results with the rolling horizon algorithms. The algorithms are tested on randomly generated problem sets as will be described in the different sections. In the last section we test the performance of the algorithms for stochastic input.

9.1 Static problems

In this section we give an experimental comparison between Model1 en Model2 for the static case (without rolling horizon). We will compare their performance with and without the logic-based Benders decomposition for different parameters (number of cranes, jobs and time interval length). We will compare both on delay and runtime. Model2 results in general in a different optimal value than Model1 due to different modeling assumptions. Model1 fixes zones for the yard cranes, while Model2 does not. On the other hand, Model2 uses a time discretization while Model1 does not. We will try to identify problem characteristics that lead to differences for the different models.

Model2 leads to a delay of a certain number of time intervals. To compare Model1 with Model2 we calculate the delay that would result from using the solution from Model2 within Model1. This is excluded in the computation times that are presented.

9.1.1 Testing settings and assumptions

The average performance of a yard crane is about 10 moves per hour. This will be our initial input for the problems under consideration. The input consists of 10 jobs per crane per hour. Half of the jobs are modeled as storage jobs and the other half of the jobs are scheduled as retrieval jobs. The input for the retrieval jobs is generated via a

uniformly random distribution among all bays. The ready times of the jobs come from a uniform integer distribution over the interval $[0, \text{Planningwindow}]$. Storage jobs are planned in sequences at the same bay. The number of jobs in a sequence is determined by again a uniform integer distribution. One to five jobs are scheduled in sequence with their ready times the handling time apart. In this way, a yard crane can stay at one bay and perform job after job without delay (unless the first one is delayed).

If for example a sequence consists of three jobs and the first is planned at time $200s$ and bay 36, the next job is planned at time $200s + 150s = 350s$ also at bay 36 (the handling time is $150s$). The third job is also planned at bay 36 with ready time $r = 500s$. For the first set of problem instances, we use a block of 40 bays per crane. The blocks are separated by 8 empty bays to enable transport of internal trucks or lane changing of the yard cranes. The yard cranes are of course not limited to their block and can move to other blocks when the workload there is high.

We test our three algorithms under different problem characteristics. To identify the influence of the length of the planning window, we create problems for four different planning windows. The longest planning period under consideration is one hour. Because a planning window of one hour will lead to large problems with many jobs (about 10 times the number of yard cranes), we only construct problems for 3 yard cranes in the lane. Planning for one hour (and fixing zones for one hour) is not preferable in practice. It is merely meant to enhance differences in the results for the different algorithms. For a planning period of half an hour, we create problems for 3 and for 6 yard cranes. More yard cranes will lead in more difficult to solve problems and as a result in long computation time. The problems with a planning window of 15 minutes or 5 minutes are more applicable as a shorter planning period will lead to better performance on objective value as well as runtime. For these planning periods we create problems for 3, 6 and 12 yard cranes. The influence of job density (number of jobs per crane per hour) and space density (number of cranes per block consisting of 40 bays) will be investigated next. We will do the same comparison for 20 bays per crane instead of 40 bays per crane. We will also investigate the effect of increasing the job density from 10 jobs per crane per hour to 15 jobs per crane per hour. For every problem characteristic, 100 different problem instances are generated.

9.1.2 Normal workload

In this subsection we will discus the results under normal workload. In Table 9.1 and Tabel 9.2 the average runtime and delay is shown for the case that jobs are distributed over 40 bays per crane and with 10 jobs per crane per hour. The results are presented for Model1 (directly solved using Gurobi), Benders (logic-based Benders decomposition for Model1) and Model2. For Model2 the heuristic restriction that jobs cannot be scheduled more than 6 time intervals (18 minutes) later than their ready time is implemented. This restriction does almost never result in different solutions, but decreases the amount of binary variables. In all tables with results presented in this section, T denotes the

planning window in minutes and m the number of cranes that are available in the lane.

Characteristics	# jobs	Model1	Benders	Model2
T=60 m=3	30	47.5068 [25]	**12.6262** [5]	47.7402 [17]
T=30 m=3	15	0.3440	**0.2073**	2.3967
T=30 m=6	30	41.6978 [26]	**2.0317**	102.5571 [72]
T=15 m=3	8	**0.0164**	0.0398	0.1140
T=15 m=6	15	0.3619	**0.2873**	7.7128
T=15 m=12	30	11.2786	**11.1816**	118.4461 [96]
T=5 m=3	3	**0.0031**	0.0062	0.0087
T=5 m=6	5	0.0663	0.0755	**0.0401**
T=5 m=12	10	**0.0961**	0.2201	1.5704

Table 9.1: Computation times in seconds of Gurobi (10 jobs per crane per hour)

Characteristics	# jobs	Model1	Benders	Model2
T=60 m=3	30	2850.9	2936.0	2993.6
T=30 m=3	15	822.9	822.9	1219.5
T=30 m=6	30	1659.4	1659.4	2882.1
T=15 m=3	8	285.8	285.8	529.0
T=15 m=6	15	542.4	542.4	1136.0
T=15 m=12	30	894.8	894.8	2858.8
T=5 m=3	3	27.0	27.0	39.0
T=5 m=6	5	52.4	52.4	314.1
T=5 m=12	10	131.3	131.3	1648.2

Table 9.2: Delay in seconds (10 jobs per crane per hour)

For some problem instances, one (or more) of the algorithms ran out of time (longer than 120s). In this case we have set the runtime to 120s and used the best found feasible solution as result for calculating the delay. This explains the difference in average delay found for the Benders decomposition and direct solving of Model1. Within brackets after the runtime we present the number of problem instances for which the algorithm ran out of time (out of 100 problem instances).

As can be seen, solving Model2 using Gurobi takes more time than solving Model1 via Benders decomposition or directly. Only in the case of a planning window of 5 minutes and scheduling 6 cranes, Model2 was solved faster. The amount of delay resulting from the found schedules was on average always in the favor of Model1. This suggest that fixing zones for the planning window leads to less delay than discretization in time. For some problem instances, for Model2 a solution with less delay was found. This is only in a few cases and on average Model1 seems to perform better. From this we can conclude that the performance of the different models is dependent on the specific

problem instance. If the planning window is large, Model2 finds delay closer to Model1. This is as expected since the zone fixing for the entire planning window is an increasing restriction for longer planning windows. The time discretization does not become worse for longer planning windows.

As we compare solving Model1 via the Benders decomposition method and directly, these experiments are not really conclusive. For the "easy" problems with very short runtime, direct solving seems to perform better on average. If the runtime increases, the Benders decomposition method seems favorable. This is in coherence with the idea that Benders decomposition is developed for large problem instances. Benders decomposition seems to work well when scheduling around 6 yard cranes. For these problem instances the Benders decomposition is solved faster with respect to solving directly than for problems with 3 or 12 yard cranes. It is not directly clear why the Benders decomposition works better for 6 cranes than for 3 or 12 cranes. In Table 9.3 we give the computation time split to different subroutines of the algorithm. We measure the mean computation times per iteration and the number of iterations that the Benders decomposition needed. We also present the amount of time spent in the master problem or in the subproblems (including heuristic lower bound procedures). Mastertime denotes the time used by Gurobi to solve the master problem. It is summed over all iterations needed, so the total time used for solving the master problem for a specific problem instance. The same holds for the quantity subtime (the computation time Gurobi uses to solve the subproblems). Only the time used by Gurobi solving the model is included in those times. The computation times for building the model in Matlab is excluded in these values.

Characteristics	iteration time	iterations	mastertime	subtime
T=15 m=3	0.0115	3.51	0.0065	0.0058
T=15 m=6	0.0440	5.74	0.1362	0.0265
T=15 m=9	0.3211	7.57	1.8628	0.0522
T=15 m=12	1.3208	7.66	9.8028	0.0844

Table 9.3: Computation times (in seconds) per iteration (Benders)

From Table 9.3 it becomes clear that the increase in runtime comes from an increase in time spent in the master problem. Apparently the master problem becomes more difficult to solve for Gurobi if the number of cranes increases. The single crane subproblems are still solved fast. It would not be surprising if the master problem computation times increase exponentially since also its worst-case runtime increases exponentially. The number of subproblems increases one to one with the number of cranes. The size of the subproblems does not increase since the number of jobs per crane remains constant. It is unknown why the computation times for the master problem are relatively short for 6 to 9 cranes compared to solving directly. The increased computation time for the very simple problems can come from building the models in Matlab or unnecessary interchanging between master problem and subproblems. The increased computation

time for many yard cranes is not easy to explain. For $T = 5$ minutes we can construct problems for even more cranes. As can be seen in Tabel 9.4 the ratio of computation times between solving directly and using the Benders decomposition does not further increase.

Characteristics	Model1	Benders	iterations
T=5 m=3	0.0031	0.0062	1.25
T=5 m=6	0.0663	0.0755	1.53
T=5 m=9	0.0322	0.0728	2.81
T=5 m=12	0.0961	0.2201	3.45
T=5 m=15	0.4031	0.7366	4.24
T=5 m=20	3.6870	4.2399	4.87
T=5 m=25	12.0393	19.8152	5.85

Table 9.4: Computation times (in seconds) for different number of cranes

The Benders decomposition results in shorter computation times when the time windows increase. For short time windows, the Benders decomposition does not seem favorable compared to solving directly. For long time intervals it is favorable based on these computation times. When the planning window increases, the subproblems concerning one single crane become harder to solve because the number of jobs per crane increases. Benders decomposition seems to profit more from solving these subproblems individually when the problem size increases. This is partly due to the lower bound procedures (although Gurobi also uses lower bound procedures in solving MILP problems), but also because the single crane problems are solved individually in a structured way. Branching on the y-variables (crane to job assignment) seems more important for this increased planning window.

9.1.3 High space density

The delay found by Model2 is dependent on the number of bays between two subsequent jobs. If jobs are close together, Model2 can perform a job in every time interval. If jobs are far apart (more than 8 bays), then Model2 needs a whole time interval to travel. This might lead to a yard crane spending 6 minutes (2 time intervals) per job. This is expected to lead to much delay because if the yard crane spends 3 minutes per job it becomes more flexible and can perform more jobs during the planning period. We do the same experiments for a planning window of 5 and 15 minutes but plan jobs over 20 bays per crane instead of 40 bays per crane. The results can be found in Table 9.5 and Table 9.6.

Compared to the case were 40 bays were used per crane, the delay found by Model2 is much closer to the delay found by Model1. The distance of 8 between to subsequent jobs is less common. Therefore, Model2 is less likely to need 2 time intervals per job instead

Characteristics	# jobs	Model1	Benders	Model2
T=15 m=3	8	**0.0223**	0.0328	0.1301
T=15 m=6	15	1.0340	**0.2444**	7.2816
T=15 m=12	30	52.0226 [31]	**20.2505** [9]	120.00 [100]
T=5 m=3	3	**0.0037**	0.0084	0.0112
T=5 m=6	5	**0.0094**	0.0172	0.0622
T=5 m=12	10	**0.1667**	0.2434	2.9238

Table 9.5: Computation times in seconds for Gurobi (high space density)

Characteristics	# jobs	Model1	Benders	Model2
T=15 m=3	8	443.4	443.4	507.5
T=15 m=6	15	914.1	914.1	973.3
T=15 m=12	30	2484.7	2523.7	2675.5
T=5 m=3	3	63.6	63.6	72.0
T=5 m=6	5	123.0	123.0	178.0
T=5 m=12	10	222.3	222.3	727.1

Table 9.6: Delay in seconds(high space density)

of one. If subsequent jobs are in the same bay (load jobs), then Model2 schedules as if it needs 3 minutes per job instead of 2.5 minutes. In a few cases this leads to a decision that results in more delay when performing the jobs according to Model1. Although schedules are made within Model2, only the sequence of jobs per crane is used. The actual schedule is based on the assumptions of Model1 to compare results. Otherwise Model2 would have trouble with the 2.5 minutes separated load jobs. This enforces the hypothesis that Model2 performs better when workload is close together. The overall delay found on average is also smaller compared to the case where jobs are far apart. So 1 crane working on 20 bays instead of 40 bays seems preferable for Model2. The separation of 8 bays does not seem to limit the schedules too much. For Model1 the opposite seems true. The delay is on average larger for the case that jobs are planned in 20 bays per crane. For Model1 the separation space seems to become a problem if jobs are scheduled too close to each other. Although the average travel distance is decreased, the amount of delay is increased due to the difficulty of dividing the lane into different zones. A separation of 8 bays is needed during the whole planning period to separate two zones. Also for a short planning window (5 minutes) the delay is increased. This means that often two jobs are too close together to be handled by different cranes.

In contrast to the delay, the runtime does not show a clear difference. Problem instances do not seem to become easier or more difficult to solve on average when changing the number of bays per crane. For $T = 5, m = 6$ the problems were actually solved in much less computation times than for the previous settings. On the other hand, for $T = 15, m = 12$ solving via Benders decomposition took twice as long as before. Solving

Model1 directly took even 5 times as long on average. For the Benders decomposition, the average number of iterations was only 3.45 (for $T = 15, m = 12$). This average number of iterations (3.45) is also relatively small compared to the previous experiment (7.66). When jobs are closer together, less separation points (8 free bays) are available. This explains the decreased number of iterations because there are less possible solutions for the master problem to try. However, the computation time used for solving the master problem was relatively long. The low number of iterations explains why the Benders decomposition works well in this case compared to direct solving. The long computation times for the master problem suggest that the assignment of cranes to jobs is quite difficult under these settings. This can also be seen from the increased amount of delay. In the previous experiment (1 crane per 40 bays) the average delay was 894.8 seconds. In this experiment the average delay is 2484.7 seconds (for solving directly). Also if we leave out the problems were no optimal solution was found in time, this delay (2021.3 seconds) is still much larger than 894.8 seconds. This also suggest that it is difficult to find a good crane to job assignment. There is a strong correlation between the optimal value (amount of delay) and the computation time used by Gurobi. Since Gurobi uses LP relaxations as lower bounds, this is probably due to the increased relative integrality gap.

9.1.4 High workload

When the workload in a certain lane of the container yard is high, yard cranes can perform more than 10 jobs per hour. In Table 9.7 and Table 9.8, the results are shown when we schedule 15 jobs per crane per hour instead of 10 jobs per crane per hour. We plan jobs in 40 bays per crane as in Table 9.1 and Table 9.2. For a planning window of 15 minutes and scheduling 12 cranes both the Benders decomposition method and Model2 were sometimes unable to return a feasible solution. Therefore we present NA (not available) instead of the average delay found.

Characteristics	# jobs	Model1	Benders	Model2
T=15 m=3	11	0.0911	**0.0835**	0.6732
T=15 m=6	23	44.1153 [21]	**1.4562**	112.7348 [85]
T=15 m=12	45	119.3479 [97]	**112.4897** [80]	120.00 [100]
T=5 m=3	4	**0.0047**	0.0145	0.0175
T=5 m=6	8	**0.0256**	0.0611	0.3540
T=5 m=12	15	**0.6835**	0.9083	27.1740 [6]

Table 9.7: Computation times in seconds for Gurobi (high workload)

The average runtime significantly increases for both models. The number of jobs (input) is increased by a factor 1.5 and the runtime is on average increased a lot more for the difficult problems. For $T = 15, m = 6$ and $T = 15, m = 12$ the computation times were almost 10 times as long as before for the Benders decomposition. For $T = 5, m = 12$

Characteristics	# jobs	Model1	Benders	Model2
T=15 m=3	11	844.0	844.0	1100.3
T=15 m=6	23	1899.4	1899.4	2972.3
T=15 m=12	45	3746.9	NA	NA
T=5 m=3	4	90.2	90.2	147.4
T=5 m=6	8	177.5	177.5	404.7
T=5 m=12	15	302.4	302.4	1283.4

Table 9.8: Delay in seconds (high workload)

the computation times were around 5 times as long. The main increase in runtime can be seen for $T = 15$ for all algorithms. For $T = 5$, the increase in computation times was less. When, as a result of the increased number of jobs per crane per hour, problems become more challenging for the algorithms, we see a more clear distinction between the different models. Model2 becomes too difficult to solve within 2 minutes for Gurobi in almost every case when scheduling for 6 jobs and a planning window of 15 minutes. If the planning window is limited to 5 minutes, both models are solved within the 2 minutes limit. There is however a clear distinction between Model1 and Model2. The Benders decomposition works well for the planning period of 15 minutes. It is much better capable of solving these large problems. For the case of a planning window of 5 minutes, direct solving seems faster than using Benders decomposition for Model1. In both cases Gurobi finds solutions well within the time limit for 12 yard cranes.

Although solving with Gurobi directly runs out of time for 22 out of 100 problem instances when planning 15 minutes ahead (6 yard cranes), it finds the same optimal value as using Benders decomposition (which was never out of time) for all except 2 problem instances. Apparently Gurobi had detected the optimal solution as being feasible but was still busy confirming it was indeed optimal. When direct solving runs out of time (for Model1 as well as Model2), in many cases it already had found the optimal solution. In other cases it had at least found a feasible solution with an objective value close to the optimal value. For the Benders decomposition this was not the case. When no optimal solution was found within the time limit, often the objective value of the best feasible solution was much worse than the optimal solution.

9.1.5 Conclusion of static case

The above experiments show that the delay coming from Model1 is on average much smaller than the delay when using Model2. Although there are cases where the opposite is true, this is far less common. On average via Model1 much less delay was found. Besides the delay found, also the computation times were in favor of Model1. Between solving Model1 directly and via the Benders decomposition, less difference was found. Both models find the same optimal value when solved within the time limit. In some cases

the Benders decomposition was solved in less computation time than solving directly, but in other cases it was the opposite. Benders decomposition seemed to produce better results for 6 to 9 cranes and a relatively long planning period (15 minutes or more).

9.2 Rolling horizon problems

In the rolling horizon approach we solve for a longer period than that we actually adopt for the definitive solution. We can remember the whole solution and use it as a starting vector for the next iteration. This can be a starting point for an optimal solution, however this need not be the case. When giving an initial solution to Gurobi, it will first try to find an optimal solution close to this starting vector. In some cases this speeds up calculations, while in other situations more computations are needed. This is the case when Gurobi tries to prove that the solution is optimal but finds out it is not. Using starting vectors as initial solution can be implemented both in the rolling horizon algorithm and the time decomposition algorithm. For the rolling horizon algorithm and the time decomposition algorithm, we have a lot of implementation options and parameter possibilities. For implementation options, there are 4 possibilities for the rolling horizon algorithm as well as the time decomposition algorithm.

1. Solve Model1 directly
2. Solve Model1 via Benders decomposition
3. Solve Model1 and remember solution as starting vector for next iteration
4. Solve Model2

In the first subsection we will focus on the first and last option for the rolling horizon and just the first for the time decomposition algorithm. After that, we will investigate the influence of using the Benders decomposition or an initial solution on the performance of the algorithms.

9.2.1 Comparison rolling horizon and time decomposition

The most important parameters for the algorithms are the time we look ahead and the time that we save per iteration. In case of the time decomposition, the time we look ahead is a multiple of the time that we save. In Section 9.1 we investigated the influence of different problem characteristics as the number of cranes, the average number of cranes per block and the number of jobs per crane per hour. In this section we use 10 jobs per crane per hour, but over a longer time window. We create problems of one hour, that are solved in parts (via rolling horizon or time decomposition). Due to this increased time, there is more fluctuation in the number of jobs available. Some time intervals will be relatively quiet while other time intervals are busy. When jobs cannot be handled on time, they will be pushed forward to the next time interval. This postponed work results

in a busy time interval for the lane. This increased amount of work can be reduced when only a few new jobs arrive over time.

The time decomposition algorithm solves the current and the next time interval several times. For this reason, solving a time interval once must be done very fast. In some cases up to 40 iterations are needed so the computation time for solving one time interval should be within a few seconds. If we want to increase the depth in the time decomposition algorithm to more than one, even shorter computation times are required. We will create problems for 3 and 6 yard cranes to investigate the capabilities of the different algorithms. In Table 9.9 we present the average results of 100 problem instances for 3 cranes. In the tables below, itermax is the maximum over the computation time per iteration. So itermax is the longest iteration time for a problem instance.

Characteristics	mean runtime	mean delay	mean itermax	max itermax
Model1-RH Tahead=5, Tsave=5	0.1266	2285.5	**0.0527**	**1.0515**
Model1-RH Tahead=10, Tsave=5	0.9319	**2063.1**	0.6119	14.0986
Model1-RH Tahead=15, Tsave=5	2.6062	2116.2	2.7977	120 [1]
Model1-RH Tahead=15, Tsave=10	1.3401	2195.2	2.4330	120 [1]
Model1-TD Tahead=5, depth=0	**0.1276**	2290.8	0.0539	1.2491
Model1-TD Tahead=5, depth=1	1.9952	2117.3	1.3920	120 [1]
Model1-TD Tahead=5, depth=2	2.6944	2090.6	1.6278	120 [1]
Model1-TD Tahead=10, depth=0	0.5277	2258.3	0.4279	14.1379
Model1-TD Tahead=10, depth=1	8.3680	2180.6	7.5000	120 [1]
Model1-TD Tahead=15, depth=0	2.8897	2439.9	2.8375	120 [1]
Model1-TD Tahead=15, depth=1	15.6043	2310.6	13.6310	120 [3]
Model2-RH Tahead=2, Tsave=2	0.8777	4183.0	0.5019	8.3828
Model2-RH Tahead=4, Tsave=2	8.7414	3285.4	6.6659	120 [3]

Table 9.9: Results over 100 problem instances of 1 hour (3 cranes, 30 jobs)

Problem instance 83 resulted in many difficulties. Because jobs were planned to close to each other, it was not possible to find a good partitioning of the jobs for the yard cranes. In a certain time interval, there must be at least 8 bays between two jobs to assign them to different zones (in the case of Model1). If these separation points do not exist for a large set of bays, the problem becomes infeasible or too hard to solve. Infeasibility occurs when 2 cranes are in a block without separation points in the next time interval. To avoid this infeasibility, we increase all ready times for the coming time period. In this case, one of the cranes has time to move away from the block and the other crane can be assigned to the jobs. This will result in much delay, but infeasibility is avoided. In the case of problem 83, it resulted still in very difficult to solve time intervals. As a result, even looking only 5 minutes ahead already resulted in an out of time report. It is possible that this specific problem instance was not representative and does not occur in practice. To get a better comparison of the algorithms for the other instances we present in Table 9.10 the results where problem instance 83 is left out. As to be expected, Model2 is less disturbed by problem instance 83. Model2 does not fix zones for an entire planning window and therefore can schedule jobs that are close together to different cranes (just not in the same time interval).

The average runtime is merely meant as an indication to the computation times used by the algorithm. It is the total computation time over all time intervals. It gives a good indication on the computational complexity, but is not directly a good measure for the applicability of the algorithm. One time interval should be solved within 2 minutes, not necessarily the whole problem instance. Therefore it is more interesting to determine the maximum computation time for one time interval (itermax). The maximum computation time per time interval is calculated for every problem instance. In Table 9.9 and Table 9.10 the average maximum computation time over all problem instances is presented. In the last column the maximum over the maximum computation times is presented. Within the brackets, the number of problem instances where the algorithm ran out of time at least once is presented.

For the rolling horizon algorithm, a longer look ahead window seems to result in better performance. At the cost of increased computation times, less delay occurs when using a larger time window. Increasing the save window results in less overall computation time, but this is mainly due to fewer time intervals. The computation time per time interval is not significantly influenced by the $Tsave$. For the time decomposition algorithm, it becomes clear that increased depth leads to less delay. In a similar way as with the rolling horizon, increasing the depth results in more computation time. So for $m = 3$ we can see a clear trade-off between runtime and delay. For these relatively small problems, the computations can be in general fast enough to solve two or three different time intervals on time. Model2 clearly finds more delay than Model1. Also the computation time of Model2 is much longer than for Model1. However, for some specific difficult problems Model2 performs better than Model1.

We did the same experiment as for 6 yard cranes instead of 3. This significantly increases the problem complexity. The computation times are much longer when scheduling 6 yard

Characteristics	mean runtime	mean delay	mean itermax	max itermax
Model1-RH Tahead=5, Tsave=5	**0.1117**	2207.9	**0.0469**	**1.0515**
Model1-RH Tahead=10, Tsave=5	0.7365	**1888.1**	0.4757	13.4364
Model1-RH Tahead=15, Tsave=5	2.7569	1949.4	1.6031	43.3896
Model1-RH Tahead=15, Tsave=10	1.5482	2000.8	1.2432	30.5453
Model1-TD Tahead=5, depth=0	0.1138	2244.4	0.0483	1.2491
Model1-TD Tahead=5, depth=1	0.4545	2042.8	0.1910	2.1546
Model1-TD Tahead=5, depth=2	1.1037	2015.9	0.4269	4.4716
Model1-TD Tahead=10, depth=0	0.3632	2211.7	0.2895	7.3701
Model1-TD Tahead=10, depth=1	3.3975	2106.4	2.6806	96.3125
Model1-TD Tahead=15, depth=0	1.6962	2368.4	1.6453	85.8136
Model1-TD Tahead=15, depth=1	12.0458	2237.7	10.1089	120 [2]
Model2-RH Tahead=2, Tsave=2	0.8620	4164.2	0.4959	8.3828
Model2-RH Tahead=4, Tsave=2	8.5467	3249.1	6.5742	120 [3]

Table 9.10: Results over 99 problem instances of 1 hour (3 cranes, 30 jobs)

cranes instead of 3 (increasing the problem size accordingly). In Table 9.11 the results are presented. Again 10 jobs per crane are scheduled over a planning window of one hour. Jobs are again distributed over 40 bays per crane.

The difference in performance for the 100 different problem instances was quite large. Some problems where solved in short computation times (and leading to small delay) while other problems resulted in many difficulties. It is difficult to determine whether these difficult problem instances reflect situation as they occur in practice. In practice schedulers can limit the number of jobs in the problem since yard cranes will not be able to perform too many jobs. To identify the differences between the algorithms for problems that can be solved well, we left out 10 problem instances that resulted in a maximum iteration time significantly larger than the average maximum iteration time.

Characteristics	mean runtime	mean delay	mean itermax	max itermax
Model1-RH Tahead=5, Tsave=5	**3.6898**	4153.6	**3.1687**	**120 [1]**
Model1-RH Tahead=10, Tsave=5	36.7235	3775.1	21.9166	120 [11]
Model1-RH Tahead=15, Tsave=5	84.0676	3678.8	39.1404	120 [21]
Model1-RH Tahead=15, Tsave=10	45.2682	**3676.9**	32.1426	120 [15]
Model1-TD Tahead=5, depth=0	4.0952	4092.9	3.5226	**120 [1]**
Model1-TD Tahead=5, depth=1	51.6208	3842.8	31.4875	120 [14]
Model1-TD Tahead=5, depth=2	89.8027	3778.9	40.7716	120 [16]
Model1-TD Tahead=10, depth=0	22.5980	4205.0	15.9049	120 [9]
Model1-TD Tahead=10, depth=1	126.3224	4074.4	75.4305	120 [39]
Model1-TD Tahead=15, depth=0	37.9899	4245.5	33.3229	120 [19]
Model1-TD Tahead=15, depth=1	181.9953	3984.8	111.3206	120 [54]
ModelLI-RH Tahead=6, Tsave=6	53.9542	11345.5	27.7851	120 [12]
ModelLI-RH Tahead=12, Tsave=6	214.1108	7555.3	86.1372	120 [56]

Table 9.11: Results over 100 problem instances of 1 hour (6 cranes, 60 jobs)

The average of the other 90 problem instances is presented in Table 9.12.

As can be seen in Table 9.12, there is still a clear trade off between computation time and delay for both the rolling horizon algorithm and the time decomposition algorithm. Increasing the depth of the time decomposition algorithm results in less delay but much longer computation times. Increasing the window of one planning period from 5 to 15 minutes result in much harder partial problems for the time decomposition algorithm. Because in many cases no optimal solution was found in time this results in worse solutions with respect to the delay. Based on these results, for the time decomposition algorithm it is best to keep the length of the intervals small. For the rolling horizon this is not the case. Looking further ahead results in less delay than looking only 5 minutes ahead. Shifting of the workload over the bays is spotted more often if $Tahead$ is larger.

Characteristics	mean runtime	mean delay	mean itermax	max itermax
Model1-RH Tahead=5, Tsave=5	0.6771	3869.4	0.4230	8.4524
Model1-RH Tahead=10, Tsave=5	15.1228	3560.3	11.8518	120 [3]
Model1-RH Tahead=15, Tsave=5	54.9853	**3380.6**	32.2385	120 [14]
Model1-RH Tahead=15, Tsave=10	30.4159	3571.0	23.9869	120 [7]
Model1-TD Tahead=5, depth=0	**0.5901**	3745.5	**0.3419**	**2.9234**
Model1-TD Tahead=5, depth=1	26.6132	3518.4	19.7704	120 [7]
Model1-TD Tahead=5, depth=2	55.6281	3422.8	28.5838	120 [8]
Model1-TD Tahead=10, depth=0	8.4624	3913.0	7.5368	120 [2]
Model1-TD Tahead=10, depth=1	98.3795	3641.9	63.2697	120 [31]
Model1-TD Tahead=15, depth=0	26.8043	3902.5	25.0432	120 [12]
Model1-TD Tahead=15, depth=1	157.2799	3667.7	99.7298	120 [44]
ModelLI-RH Tahead=6, Tsave=6	36.8658	11082.1	20.8716	120 [5]
ModelLI-RH Tahead=12, Tsave=6	196.3058	7261.7	82.4968	120 [46]

Table 9.12: Results over 90 problem instances of 1 hour (6 cranes, 60 jobs)

For these problem instances it is more valuable to look far ahead in spite of fixing the zones for a longer period. Updating the zones after 5 minutes instead of 10 minutes results in slightly less delay for these problem instances.

The time decomposition algorithm works worse when the number of cranes increases. The time decomposition then needs many iterations instead of only a few. When there are more cranes, there are more possibilities to alter the solution for the first time period. There are many variations possible for the first time period that might result in less delay in the second time period. This increase in iterations seems to grow fast when increasing the number of cranes. For the problems corresponding with Table 9.9 (3 cranes) the decomposition algorithms needed around 6 iterations on average. For the problems corresponding to Table 9.11 this was already 30 iterations. This time

decomposition method might therefore not be very applicable for terminals with long lanes were many yard cranes work simultaneously.

9.2.2 Different implementations

In this section we compare three different algorithms that we can using for solving Model1 in the rolling horizon framework. The partial problem within the rolling horizon are solved with three different methods. Direct means we solve the partial problems directly using Gurobi as in the previous subsection. The memory algorithms solves the partial problems by using an initial solution from the previous partial problem if available. The Benders decomposition algorithm solves the partial problems using the Benders decompostion. We present again the average results over 100 problem instances of 1 hour. In Table 9.13 we present the results for problems with 3 cranes (and 30 jobs in one hour). In Table 9.13 the results for 6 cranes (60 jobs) are presented. Itermax is again the longest iteration runtime of a problem instance. Itermax2 is the maximum over the number of iterations Benders decomposition needed over the time intervals.

Algorithm	Tahead=5 Tsave=5	Tahead=10 Tsave=5	Tahead=15 Tsave=5	Tahead=15 Tsave=15
Runtime direct	**0.1352**	1.2281	6.8493	2.9573
Runtime memory	0.1524	**1.1105**	6.9502	2.9332
Runtime Benders	0.4518	1.8197	**3.1242**	**1.7465**
Delay direct	2235.7	2004.6	2064.2	2128.1
Delay memory	2235.7	2004.6	2064.2	2128.1
Delay Benders	2238.3	2020.1	2025.5	2124.4
Mean itermax direct	**0.0625**	0.7162	2.9098	2.4242
Mean itermax memory	0.0708	**0.6170**	3.0424	2.4500
Mean itermax Benders	0.2586	0.9172	**1.5179**	**1.2039**
Max itermax direct	**1.4098**	19.3374	120.9532	96.8614
Max itermax memory	1.4786	**14.3877**	120.9557	109.0934
Max itermax Benders	18.2585	33.5971	**40.9781**	**35.5787**
Mean itermax2 Benders	4.9600	6.1400	7.6800	7.2700

Table 9.13: Results over 100 problem instances of 1 hour (3 cranes, 30 jobs)

The delay found by the different algorithms is very close together. This is as expected since the algorithms should find the same optimal value for the same partial problem within the rolling horizon. The small differences between the algorithms follow from two different factors. First, for some problem instances one or more of the algorithms ran out of time (120 seconds or 10 seconds for a subproblem in the Benders decomposition) for a partial problem. The best found feasible solution is then returned which is not necessarily the optimal solution. As in the static problems, Benders decomposition often did return

Algorithm	Tahead=5 Tsave=5	Tahead=10 Tsave=5	Tahead=15 Tsave=5	Tahead=15 Tsave=15
Runtime direct	3.3348	38.5151	82.3440	47.0435
Runtime memory	3.5855	36.8953	84.8018	47.1374
Runtime Benders	1.5005	5.8288	15.1964	7.1031
Delay direct	4137.0	3877.2	3731.4	3929.7
Delay memory	4137.0	3877.2	3729.2	3929.9
Delay Benders	4068.6	3858.6	3754.6	3998.5
Mean itermax direct	2.8263	21.6719	37.9084	31.9577
Mean itermax memory	2.9475	20.3833	37.5642	33.0503
Mean itermax Benders	0.8970	3.3064	7.0230	4.8958
Max itermax direct	120.2786	120.6164	120.8052	120.4588
Max itermax memory	120.3028	120.8463	121.2336	120.4288
Max itermax Benders	20.1415	50.6021	81.5588	70.9712
Mean itermax2 Benders	8.6500	12.1700	15.1800	13.7700

Table 9.14: Results over 100 problem instances of 1 hour (6 cranes, 60 jobs)

worse solutions than direct solving when Gurobi ran out of time. The second factor that leads to slightly different results is that in some partial problems the different algorithms choose a different optimal solution. In many cases the optimal solution is not unique and the different algorithms can choose a different solution. This influences the next partial problem since the starting position of the cranes is different.

If we compare the different lengths of time periods, we see the same results as in the previous section. For 3 cranes, it seems most advantageous to look 10 minutes ahead. For 6 cranes it is more advantageous to look 15 minutes ahead (based on the delay for the schedules). In both cases, it is more advantageous to update the schedule every 5 minutes above every 10 minutes.

In the computation times we see a clear difference between using Benders decomposition and direct solving. The difference in computation time between using an initial solution and not using an initial solution is not so clear. Remembering part of the previous schedule does not seem to help much. For some problem instances it helped a little, but for other problem instances the computation times were even a bit worse. The Benders decomposition method was for 6 cranes more than 6 times faster than solving Model1 directly or with the aid of the memory algorithm (except for Tahead=5, Tsave=5). For the problems with only 3 cranes, the difference was less clear. For the longer time periods (Tahead=15) the Benders decomposition resulted in smaller computation times. For the small time periods (Tahead=5,10) the Benders decomposition resulted in longer computation times. This is in comparison with the static results. Also for the static results, Benders decomposition seemed to work best for 6 cranes and planning windows of at least 15 minutes. It enforces the idea that Benders decomposition works better for

problem instances with a larger input size.

If we look at the maximum computation time needed for one iteration of the rolling horizon for a problem instance, the results are also in favor of the Benders decomposition. Especially for the problem instances with 6 cranes, itermax was much smaller for the Benders decomposition than for the direct or memory algorithm. The number of iterations needed by the Benders decomposition corresponded to the average computation times. More iterations resulted in longer computation times. This suggest that the influence of solving a difficult subproblem (and thus the time needed for one iteration) was small. This can be a consequence of the time limit of 10 seconds for solving a specific subproblem.

9.3 Stochastic results

In this section we discus some experimental results for using the stochastic input. We tested the performance of the algorithm based on expectations as well as the algorithm based on simulation. A drawback of the simulation algorithm is that it requires more computation time because it has to solve the second problem in an iteration several times. Let N denote the number of simulations done in the algorithm. The simulation algorithm has to solve $N+1$ problems in each iteration of the time decomposition algorithm. The expected value algorithm has to solve only 2 problems in each iteration (as is in the deterministic case). To test the performance of both algorithms, we compare the delay that is found by those algorithms with the delay that would be found if the exact ready times were known. First the algorithms were implemented in a rolling horizon version as in Section 9.2. The results of 50 problem instances of 1 hour, 3 cranes and $\lambda = 100$ seconds are presented in Table 9.15.

	Exact	Expectation	Simulation (N=5)
mean Runtime	0.7275	0.7254	48.2108
max Runtime	2.0683	2.2664	513.8702
min Runtime	0.3539	0.4290	5.1360
mean Delay	3380.5421	3384.3454	3452.9225
max Delay	4639.4645	4766.3088	4908.6092
min Delay	1969.3200	1969.3200	1969.3200

Table 9.15: Average results of 50 problem instances of 1 hour with 10 jobs. $m = 3$, $\lambda = 100s$ and Tahead= $360s$

As in the previous section, results of a certain time period influences the next time period. Therefore the situation occurred that the expectation algorithm or the simulation algorithm found a better solution than the exact solution over a whole hour. Therefore

we study the effects of the stochastic algorithms also for static problems (without the rolling horizon). We plan the arrival time at the gate for the coming 20 minutes. The lateness of these jobs is modeled by an exponential distribution with mean $\lambda = 180s$. In the static case we have less influence of the start position of the cranes and do not have postponed workload from a previous time period. Therefore problems are rather easy to solve and solutions might be trivial. This leads to very little to no difference between the exact solution and the solutions of the stochastic algorithms. To increase the difficulty of the problems and therefore the effects of the stochastic input, we schedule 10 jobs per crane per hour and over 20 bays per crane. This is equal to the high space density of Section 9.1. Because we limit our problem instances to 3 cranes, the problem instances can still be solved reasonable fast, even for 20 minutes. Because we want to decompose the planning window of 20 minutes into two time periods, we assume the input is deterministic for the first 13 minutes. So jobs that are ready within 13 minutes are assumed deterministic and are solved in the first time period, while the others are stochastic and are solved in the second time period. After determining the solution for the first time period, the second time period is solved with the exact arrival times. The results are presented in Table 9.16.

	Exact	Expectation	Simulation (N=1)	Simulation (N=10)	Simulation (N=50)
mean Runtime	0.0651	0.0618	0.0607	0.7752	3.5522
mean Delay	514.9835	528.2001	526.3749	521.8967	521.8967
# Optimal	100	91	91	94	94

Table 9.16: Average results of 100 problem instances of 20 minutes with 10 jobs. $m = 3$, $\lambda = 180s$ and Tahead= $780s$ (10 jobs per crane per hour)

For these problem instances the simulation algorithm gave better results than the expectation algorithm. This is contrast to the problem instances that were solved using the rolling horizon. This is probably due to the difference of a static problem instance or the rolling horizon problem instances. For the rolling horizon instances it is important that cranes do not all go to one side of the lane since that might result in bad solutions later on. Using the expected arrival times of jobs gives a more equal distribution of jobs over time, while simulation (and the real realization) might give a more capricious distribution of the jobs. Apparently, following the expected arrival times keeps the cranes more equally distributed over the lane than following simulated arrival times. Doing the same experiment on the same problem instances gave the same result (using different simulated values). Also doing the same experiment on different problem instances but with the same characteristics resulted in similar results (see Table 9.17).

The computation times clearly increase as N increases. This is to be expected because in a certain iteration, The second problem has to be solved N times instead of once. The mean delay found for these problem instances decreases for an increasing N. So the increased number of simulations result in better solutions at the cost of an increase

	Exact	Expectation	Simulation (N=1)	Simulation (N=10)	Simulation (N=50)
mean Runtime	0.0962	0.0906	0.0930	0.7959	4.0587
mean Delay	534.9364	564.6926	556.3802	553.4258	541.0444
# Optimal	100	88	92	95	96

Table 9.17: Average results of 100 problem instances of 20 minutes with 10 jobs. $m = 3$, $\lambda = 180s$ and Tahead= $780s$ (10 jobs per crane per hour)

in computation times. The different simulations can be done in parallel to reduce total computation times if multiple processors are available.

We did another experiment were the job density is increased to 12 jobs per crane per hour. We also adjusted the parameter of the exponential distribution of the job lateness to $\lambda = 300s$. The results are presented in Table 9.18. The results are not very different to the results of Table 9.16 and Table 9.17. The algorithm using expected values seems to perform slightly worse compared to the algorithm using simulation. A larger parameter λ for the exponential distribution results in a higher variation of the lateness of jobs. Scheduling on the expected value seems less appropriate.

	Exact	Expectation	Simulation (N=1)	Simulation (N=10)	Simulation (N=50)
mean Runtime	0.1176	0.1515	0.1322	1.1455	5.5844
mean Delay	691.6793	732.6055	717.6559	713.1076	702.0474
# Optimal	100	86	87	87	90

Table 9.18: Average results of 100 problem instances of 20 minutes with 12 jobs. $m = 3$, $\lambda = 300s$ and Tahead= $780s$ (12 jobs per crane per hour)

As we have seen in Section 9.2 the time decomposition did not work well when we increased the number of cranes. We made 100 problem instances for 6 cranes, and tested the algorithms also on these problem instances. As expected, the number of iterations again drastically increased because of many more possible job to crane assignments in the first time period. In Table 9.19 the results are shown. They are in comparison with the results above, except for the increase of N for the simulation algorithm. In these problem instances the simulation algorithm ran out of time many times. This shows that using many simulations for solving the problem is a serious drawback. The extra simulations resulted in less iterations for the time decomposition algorithm and therefore worse solutions. This drawback can be reduced by increasing the computational capacity.

	Exact	Expectation	Simulation (N=1)	Simulation (N=10)	Simulation (N=50)
mean Runtime	7.3873	7.6036	6.6581	43.4587	96.2972
mean Delay	1.3081	1.3890	1.3423	1.3483	1.3634
# Optimal	100	81	86	86	84

Table 9.19: Average results of 100 problem instances of 20 minutes with 24 jobs. $m = 6$, $\lambda = 300s$ and Tahead= $780s$ (12 jobs per crane per hour)

Chapter 10

Conclusion

The main goal of this research was to find a good MILP formulation for the *yard crane scheduling problem*. By a good MILP formulation we mean that the resulting problem is solvable within a few minutes and returns good schedules. We have tested different models and algorithms and investigated their applicability using Gurobi for solving the MILP problems.

Model1 performed best under our testing assumptions. In this model, non-overlapping zones are assigned to the yard cranes. The completion time of jobs is modeled by continuous variables. Model2 is based on the best found MILP formulation in literature. A time discretization is used to model the completion times. In contrast to Model1, no zones are assigned to the yard cranes. A yard crane is assigned to a job and non-crossing is enforced per time interval (in contrast to Model1 where it is an entire planning period). Model1 found schedules with less delay in shorter computation times. Fixing zones for the yard cranes seems to result in better models than time discretization. Because the yard crane scheduling problem is NP-hard, the input size should not be to large. Planning for up to 15 minutes ahead with 6 cranes is in general solvable within 2 minutes. If different lanes are solved in parallel, this means that if lanes are limited to 6 cranes we can schedule 15 minutes ahead. If lanes can contain up to 12 cranes, it is possible to schedule 5 minutes ahead. If more computational power would be available (compared to the quad core 3.40 GHz, 8 GB RAM computer we used in our experiments) slightly larger problems could be solved within 2 minutes.

Logic-based Benders decomposition was introduced as an attempt to further reduce the computation times of Gurobi for solving Model1. Benders decomposition split the yard crane scheduling problem in a master problem determining the zones for the yard cranes and a subproblem for each zone where the exact trajectory of the yard crane via the different jobs is calculated. Iterating between the master problem and the subproblems results in more problems to solve, but with a smaller input size each. Since the input size grows exponentially with the number of jobs, solving multiple smaller problems can be advantageous to solving one larger problem. It was found that, depending on the

input size, solving via Benders decomposition can indeed reduce the computation times. The reduction in computation time was most significant for a relatively long planning window (30 minutes). For a short planning window (5 minutes), there was no reduction in computation time.

Inspired by the Benders decomposition, also a time decomposition was presented. Instead of creating a schedule for the entire planning window, the planning window was divided into multiple time periods. These time periods were solved separately, again with the aim to reduce the input size. The end position of cranes in a time period influences the next time period. The different time periods are solved iteratively such that the delay over all time periods is minimized. Compared to solving directly, solving multiple time periods separately has the advantage that zones can be redefined each time period and are not fixed for the entire planning window. A drawback is that the jobs cannot be interchanged in sequence between different time periods. It might be advantageous to perform the first job of time period 2 before the last job of time period 1 is performed, depending on the job locations. This time decomposition worked well when only a few cranes were available for a problem instance. When the number of cranes increased, however, the time decomposition was found to perform poorly. This is caused by the fact that there are too many opportunities to change the job to crane assignment, resulting in many iterations. On the other hand, the time decomposition generally leads to solutions with less delay because the zones are fixed for shorter time intervals.

The exact arrival time of jobs in the container yard is very uncertain. Only on short notice, the arrival times are well predictable. The time decomposition creates a nice framework to include stochastic arrival times. The different time periods allow for distinction in deterministic or stochastic ready times. We can assume that in the first time period (for example, 5 minutes) job arrival times are deterministic while the next time interval job arrival times are uncertain. Here the assumption was made that job arrival times follow an exponential distribution. We presented two different algorithms that can be used to solve problems that are modeled in this manner. The first algorithm uses expected arrival times for the uncertain job arrival times. The second algorithm uses simulations to determine a good schedule under different realizations of the uncertain job arrival times. For fixed time windows the simulation algorithm was found to perform better than the algorithm based on expected arrival times. When the number of simulations is increased, the schedules obtained by the algorithm are better. These schedules result in less delay for the yard cranes. On the other hand, the computation times increase with the number of simulations.

In general we can conclude that for planning windows longer than 15 minutes it is advisable to use the Benders decomposition to solve the yard crane scheduling problem. For shorter planning windows, direct solving with Gurobi seems more applicable. Time decomposition shows good results and is advisable when the number of cranes is limited. If the number of cranes is 6 or more, the time decomposition in its present form does not lead to good schedules as a consequence of too many iterations.

Chapter 11

Discussion

11.1 Validation

For modeling the *yard crane scheduling problem*, it is necessary to make certain assumptions. However, it appears that these assumptions are not straightforward, since both in literature as well as in practice these assumptions can differ a lot. For example, the separation space between yard cranes and the time it takes for a yard crane to perform a job are quite different. Also operational choices can be different. In some terminals yard cranes need to load containers from a certain group, while for other terminals the yard cranes are assigned to specific containers. This research was conducted at TBA and their scheduler is used for specific crane to container assignment. The scheduler should work for multiple terminals and therefore some parts need to be adjustable. The assumptions that were made in the thesis are mostly done such that they are compatible with their yard crane scheduler.

It is difficult to precisely comment on the integration of the research results in the existing software, as implementation can be done on various levels. The lowest level of implementation would be to only use the obtained MILP formulations. For the TBA yard crane scheduler, the MILP could be used to determine the zones assigned to the yard cranes. Because of this, Model1 is most suited for this, since solutions for this MILP result in the smallest delay and non-overlapping zones. At the moment, non-overlapping zones are used in the scheduler. Because fixing zones can lead to worse schedules compared to not fixing zones, zone fixing should be done for short time intervals. For this reason as well as the computation times, it is advisable to use short planning windows and update the solution often. As a consequence, this would reduce the applicability of the Benders decomposition.

The testing and evaluation of the performance of the algorithms is done via problems that were created randomly. Although those problem instances are created based on advice of experts from TBA and parameters they provided, they are not real world

problem instances. They reflect real situations, but are usually deterministic while in real terminals changes in planning occur more frequent. However, the algorithms performed well on these test cases. Some specific problem instances led to problems, but some extensions to the algorithm to deal with these unwelcome events might reduce this problem. The main difference between the problem instances used for testing and real world application is the uncertainty of job arrival times. When jobs arrive at an unexpected moment or do not arrive at the expected arrival time, the schedules are less useful. Last minute changes are not unusual in container terminals. Because of this, the algorithms need further development to deal with this last minute uncertainty.

The main difficulties for the algorithms were if one crane has to perform much more jobs than it can technically handle in a short time period or when jobs were scheduled such that no good separation space was available for the yard cranes. These problems can be worked around when an alternative heuristic is used for difficult problems. The separation constraint can be discarded when no good separation space is available. Nowadays yard cranes (RTG cranes) are manned and they can avoid coming too close to each other themselves. When RTG cranes become automated, this will not be possible. In that case a heuristic should be developed to temporarily skip one job such that a separation space becomes available. Another solution would be to create separation space by the optimization problem that determines when containers are needed from or to the yard. A bay can be changed for a storage job or a load job can be postponed or advanced to create a separation space.

11.2 Future research

There are three main issues that require more research to improve the applicability of the presented algorithms. First of all more research is needed to further involve uncertainty in the current models. Not only uncertainty for the arrival of trucks at the gate, but also uncertainty for the travel time of trucks within the yard. Currently this is the most important missing element in research for the yard crane scheduling. Good schedules are not applicable when jobs arrive late or early.

A second improvement would be some additional heuristics that can be used in specific cases. In some problem instances too many jobs were planned to solve with the current algorithms. In these cases a heuristic solution can be used or another alternative for the algorithm is needed. Another possible improvement would be to define an MILP formulation were no zone fixing and no time interval fixing is done. In this case optimal solutions without the limitation to separate zones or discrete time intervals could be found. The MILP formulation should still be solvable fast enough to be applicable.

Although not directly useful for the current yard crane scheduler from TBA, an integrated approach for the storage plan and yard crane scheduling would be useful. From our experiments it follows that the possibility to find good schedules for the RTG cranes

is very dependent on the position and ready times of the jobs that need to be performed. A good distribution of these jobs ensures optimal schedules that are easy to find. In this case, not the delay coming from the yard cranes arriving by the jobs has to be optimized. The main goal for the yard of a container terminal is to ensure that the quay crane can perform its jobs on time. Integrating a storage plan for the yard, the yard crane scheduling and even the vehicle routing between yard cranes and quay cranes will make this objective easier to pursue. Creating such an integrated model that can be solved in limited time seems challenging. However (or maybe just because it is challenging), more research to an integrated approach is encouraged.

Bibliography

Tolga Bektaş. Formulations and Benders decomposition algorithms for multidepot salesmen problems with load balancing. *European Journal of Operational Research*, 216 (1):83–93, 2012.

Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

Christian Bierwirth and Frank Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, 2010.

Mernout Burger. Exact and compact formulation of the fixed-destination travelling salesman problem by cycle imposement through node currents. In *Operations Research Proceedings 2013*, pages 83–88. Springer, 2014.

Jin Xin Cao, Der-Horng Lee, Jiang Hang Chen, and Qixin Shi. The integrated yard truck and yard crane scheduling problem: Benders' decomposition-based methods. *Transportation Research Part E: Logistics and Transportation Review*, 46(3):344–353, 2010.

Enrique Castillo, R Minguez, AJ Conejo, and R Garcia-Bertrand. *Decomposition techniques in mathematical programming*. Springer, 2006.

Daofang Chang, Zuhua Jiang, Wei Yan, and Junliang He. Developing a dynamic rolling-horizon decision strategy for yard crane scheduling. *Advanced Engineering Informatics*, 25(3):485–494, 2011.

Lu Chen, Nathalie Bostel, Pierre Dejax, Jianguo Cai, and Lifeng Xi. A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research*, 181(1):40–58, 2007.

Raymond K Cheung, Chung-Lun Li, and Wuqin Lin. Interblock crane deployment in container terminals. *Transportation Science*, 36(1):79–93, 2002.

Yingyi Chu and Quanshi Xia. Generating Benders cuts for a general class of integer programming problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 127–141. Springer, 2004.

Gianni Codato and Matteo Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

Carlos F Daganzo. The crane scheduling problem. *Transportation Research Part B: Methodological*, 23(3):159–175, 1989.

Xi Guo and Shell Ying Huang. Dynamic space and time partitioning for yard crane workload management in container terminals. *Transportation Science*, 46(1):134–148, 2012.

Xi Guo, Shell Ying Huang, Wen Jing Hsu, and Malcolm Yoke Hean Low. Dynamic yard crane dispatching in container terminals with predicted vehicle arrival information. *Advanced Engineering Informatics*, 25(3):472–484, 2011.

Junliang He, Daofang Chang, Weijian Mi, and Wei Yan. A hybrid parallel genetic algorithm for yard crane scheduling. *Transportation Research Part E: Logistics and Transportation Review*, 46(1):136–155, 2010.

John N Hooker and Greger Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

Kap Hwan Kim and Hong Bae Kim. Segregating space allocation models for container inventories in port container terminals. *International Journal of Production Economics*, 59(1):415–423, 1999.

Akio Imai, Etsuko Nishimura, and Stratos Papadimitriou. The dynamic berth allocation problem for a container port. *Transportation Research Part B: Methodological*, 35(4): 401–417, 2001.

H Javanshir and SR Seyedalizadeh Ganji. Yard crane scheduling in port container terminals using genetic algorithm. *Journal of Industrial Engineering International*, 6(11):39–50, 2010.

Sung Ho Jung and Kap Hwan Kim. Load scheduling for multiple quay cranes in port container terminals. *Journal of Intelligent Manufacturing*, 17(4):479–492, 2006.

Kap Hwan Kim and Jong Wook Bae. A look-ahead dispatching method for automated guided vehicles in automated port container terminals. *Transportation Science*, 38(2): 224–234, 2004.

Kap Hwan Kim and Ki Young Kim. An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science*, 33(1):17–33, 1999.

Kap Hwan Kim and Young-Man Park. A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156(3):752–768, 2004.

Ki Young Kim and Kap Hwan Kim. Heuristic algorithms for routing yard-side equipment for minimizing loading times in container terminals. *Naval Research Logistics (NRL)*, 50(5):498–514, 2003.

Der-Horng Lee, Zhi Cao, and Qiang Meng. Scheduling of two-transtainer systems for loading outbound containers in port container terminals with simulated annealing algorithm. *International Journal of Production Economics*, 107(1):115–124, 2007.

Der-Horng Lee, Hui Qiu Wang, and Lixin Miao. Quay crane scheduling with non-interference constraints in port container terminals. *Transportation Research Part E: Logistics and Transportation Review*, 44(1):124–135, 2008.

Der-Horng Lee, Jin Xin Cao, Qixin Shi, and Jiang Hang Chen. A heuristic algorithm for yard truck scheduling and storage allocation problems. *Transportation Research Part E: Logistics and Transportation Review*, 45(5):810–820, 2009.

Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

Wenkai Li, Yong Wu, Matthew EH Petering, Mark Goh, and Robert de Souza. Discrete time model and algorithms for container yard crane scheduling. *European Journal of Operational Research*, 198(1):165–172, 2009.

Andrew Lim. The berth planning problem. *Operations Research Letters*, 22(2):105–110, 1998.

Andrew Lim, Brian Rodrigues, and Zhou Xu. Approximation schemes for the crane scheduling problem. In *Algorithm theory-swat 2004*, pages 323–335. Springer, 2004.

Andrew Lim, Brian Rodrigues, and Zhou Xu. A m-parallel crane scheduling problem with a non-crossing constraint. *Naval Research Logistics (NRL)*, 54(2):115–127, 2007.

Richard Linn, Ji-yin Liu, Yat-wah Wan, Chuqian Zhang, and Katta G Murty. Rubber tired gantry crane deployment for container yard operation. *Computers & Industrial Engineering*, 45(3):429–442, 2003.

Richard J Linn and Chu-Qian Zhang. A heuristic for dynamic yard crane deployment in a container terminal. *IIE Transactions*, 35(2):161–174, 2003.

KL Mak and D Sun. Scheduling yard cranes in a container terminal using a new genetic approach. *Engineering Letters*, 17(4):274, 2009.

Patrick JM Meersmans and Rommert Dekker. Operations research supports container handling. Technical report, Econometric Institute Research Papers, 2001.

Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

Katta G Murty, Jiyin Liu, Yat-wah Wan, and Richard Linn. A decision support system for operations in a container terminal. *Decision Support Systems*, 39(3):309–332, 2005.

Ananthapadmanabhan Narasimhan and Udatta S Palekar. Analysis and algorithms for the transtainer routing problem in container port operations. *Transportation Science*, 36(1):63–78, 2002.

WC Ng. Crane scheduling in container yards with inter-crane interference. *European Journal of Operational Research*, 164(1):64–78, 2005.

WC Ng and KL Mak. Yard crane scheduling in port container terminals. *Applied Mathematical Modelling*, 29(3):263–276, 2005.

Matthew EH Petering and Katta G Murty. Simulation analysis of algorithms for container storage and yard crane scheduling at a container terminal. In *Proceedings of the Second International Intelligent Logistics Systems Conference, Brisbane, Australia*, pages 19–1, 2006.

Matthew EH Petering, Yong Wu, Wenkai Li, Mark Goh, and Robert de Souza. Development and simulation analysis of real-time yard crane control systems for seaport container transshipment terminals. *OR Spectrum*, 31(4):801–835, 2009.

Sartaj K Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*, 23(1):116–127, 1976.

Riazi Sarmad, Oskar Wigström, Seatzu Carla, and Bengt Lennartson. Benders/gossip methods for heterogeneous multi-vehicle routing problems. In *Proc. 18th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2013), Cagliari, September*, 2013.

John E Savage. Models of computation. *Exploring the Power of Computing*, 1998.

Omor Sharif, Nathan Huynh, Mashrur Chowdhury, and Jose M Vidal. An agent-based solution framework for inter-block yard crane scheduling problems. *International Journal of Transportation Science and Technology*, 1(2):109–130, 2012.

Steven E Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media, 2004.

Robert Stahlbock and Stefan Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

Dirk Steenken, Stefan Voß, and Robert Stahlbock. Container terminal operation and operations research-a classification and literature review. *OR Spectrum*, 26(1):3–49, 2004.

Ilaria Vacca, Michel Bierlaire, and Matteo Salani. Optimization at container terminals: status, trends and perspectives. In *Proc. of Swiss Transport Research Conference*. Citeseer, 2007.

Iris FA Vis and Rene De Koster. Transshipment of containers at a container terminal: An overview. *European Journal of Operational Research*, 147(1):1–16, 2003.

Bin Wang and Tao Yang. Multi-objective and stochastic optimization model of transit containers storage in a transshipment port yard. *Applied Mechanics and Materials*, 411:2680–2683, 2013.

Chuqian Zhang, Yat-wah Wan, Jiyin Liu, and Richard J Linn. Dynamic crane deployment in container storage yards. *Transportation Research Part B: Methodological*, 36 (6):537–555, 2002.

Appendix A

Additional binary variables

We investigated the introduction of additional variables describing the direct preceeder of a job. The x variables in Model1 describe which jobs preceed job i within its zone. But they include all jobs that preceed job i, not only the direct preceeder. In some researches it is advertised that binary variables describing direct preceders result in better models. Below we introduce extra variables z that describe these direct preceders.

The new binary decision variables z_{ij} are defined as follows:

$$z_{ij} = \begin{cases} 1 & \text{if job } i \text{ precedes job } j \text{ directly in a zone;} \\ 0 & \text{otherwise.} \end{cases}$$

These variables are very dependent on the variables x_{ij}. To define them well, we also need some variables for the first and last job in a zone. So we define:

$$z_{i0} = \begin{cases} 1 & \text{if job } i \text{ is the last job in a zone;} \\ 0 & \text{otherwise.} \end{cases}$$
$$z_{0j} = \begin{cases} 1 & \text{if job } j \text{ is the first job in a zone;} \\ 0 & \text{otherwise.} \end{cases}$$

This means that in total we introduce $n(n-1)+2n = n(n+1)$ new decision variables. We also add $n(n-1)+2n+2$ new constraints which makes a total of $(3+m)n^2+(1-m)n+2$

constraints.

$$z_{ij} \leq x_{ij} \qquad i,j = 1..n; i \neq j \tag{A.1}$$

$$\sum_{i=1}^{n} z_{ij} + z_{0j} = 1 \qquad i = 1..n \tag{A.2}$$

$$\sum_{j=1}^{n} z_{0j} \leq m \tag{A.3}$$

$$\sum_{j=1}^{n} z_{ij} + z_{i0} = 1 \qquad i = 1..n \tag{A.4}$$

$$\sum_{i=1}^{n} z_{i0} \leq m \tag{A.5}$$

$$x_{ij}, y_{ik}, z_{ij}, z_{0j}, z_{i0} \in \{0,1\} \qquad i,j = 1..n; i \neq j; k = 1..m \tag{A.6}$$

This addition of variables has a bad effect on the performance. The runtime of Gurobi on this model is longer than the runtime of Gurobi on Model1. For one out of four problem instances, it took even a factor 10 longer to find an optimal solution. However, it might be useful in combination with other adjustments. It is possible to add additional constraints to create better LP-relaxations with the aid of these new variables. We at least see some difference in the LP solution. Instead of splitting jobs over two cranes, the LP-solution assigns all jobs to the rightmost crane. The x-variables are no longer zero, but consists of numbers around 0.9 beneath the diagonal and 0.1 above the diagonal. Different values are possible. Constraint (A.2) forces $z_{ij} > 0$ for some entries. Constraint (A.1) passes this forward to x_{ij}. Since $x_{ij} < 1$ in most cases and M is a very big number, Constraints (A.5) still do not become active.

Example: Assume we have the very simple example with 1 crane and 2 jobs and $r_1 = 10, r_2 = 30, h_1 = 10, h_2 = 10, b_1 = 5$ and $b_2 = 10$. We therefore have $d_{ij} = d_{ji} = |b_i - b_j| = 5$ and $M = \max_{i=1..n}(r) + n(\max_{i,j=1..n}(d) + \max_{i=1..n}(h)) = 30 + 2(5+10) = 60$. If we discard the binary constraints, $z_{12} = \frac{2}{3}, z_{21} = \frac{1}{3}, z_{10} = \frac{1}{3}, z_{20} = \frac{2}{3}, z_{01} = \frac{2}{3}$ and $z_{02} = \frac{1}{3}$ is allowed by (A.2), (A.3), (A.4) and (A.5). Constraints (A.1) restricts $x_{12} \geq \frac{2}{3}$ and $x_{21} \geq \frac{1}{3}$. If we fix $x_{12} = \frac{2}{3}$ and $x_{21} = \frac{1}{3}$ we get the following constraints for t: $t_2 - t_1 \geq d_{12} + h_2 - (1 - \frac{2}{3})M = 5 + 10 - \frac{1}{3} \cdot 60 = -5$ and $t_1 - t_2 \geq d_{21} + h_1 - (1 - \frac{1}{3})M = 5 + 10 - \frac{2}{3} \cdot 60 = -25$. We are allowed to choose $t_1 = r_1 + h_1 = 20$ and $t_2 = r_2 + h_2 = 40$, resulting in zero delay. Something similar happens for larger problem instances.

Appendix B

Benders decomposition methods

In this appendix we describe in more detail the Benders decomposition method and apply it to Model1 and model3. At first we (again) introduce the classical Benders decomposition method. We give two different classical Benders cuts. One based on the dual objective and one based on the primal objective of the subproblems. We apply these methods to Model1 and Model3. Thereafter we introduce a combinatorial Benders cut and a relaxed CGPr-cut. We apply these on Model1 only.

B.1 Classical Benders decomposition

Benders decomposition exploits a block structure in the linear LP/MILP problem formulation to decompose the problem into several smaller problems. Subproblems need to be solved fast and lead to new constraints for the master problem. In a typical iteration, first the master problem is solved. This fixes some values that are used as input for the subproblems. The subproblems are solved using the values of the variables in the master problem. These results are translated to a Benders cut (defined later) which should restrict the MP for the next iteration (delayed row-generation). The objective function might depend on the variables in the subproblem, the variables in the master problem or both. Depending on this, different cut-generation methods are needed. Assume we have the following general problem, where y is a vector of binary variables and $x_1, \ldots, x_k$ vectors of continuous variables:

$$
\begin{array}{lrcrcrcccrcl}
\text{minimize} & \mathbf{c}^T\mathbf{y} & + & \mathbf{f}_1^T\mathbf{x}_1 & + & \mathbf{f}_2^T\mathbf{x}_2 & + & \cdots & + & \mathbf{f}_k^T\mathbf{x}_k & & \\
\text{subject to} & \mathbf{A}\mathbf{y} & & & & & & & & & = & \mathbf{b} \\
 & \mathbf{B}_1\mathbf{y} & + & \mathbf{D}_1\mathbf{x}_1 & & & & & & & = & \mathbf{d}_1 \\
 & \mathbf{B}_2\mathbf{y} & & & + & \mathbf{D}_2\mathbf{x}_2 & & & & & = & \mathbf{d}_2 \\
 & \vdots & & & & & & \ddots & & & \vdots & \\
 & \mathbf{B}_k\mathbf{y} & & & & & & & + & \mathbf{D}_k\mathbf{x}_k & = & \mathbf{d}_k
\end{array}
$$

$$\mathbf{y} \in \{0,1\} \quad \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \geq 0$$

In this problem $\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_k, \mathbf{D}_1, \ldots, \mathbf{D}_k$ are constraint matrices, $\mathbf{c}, \mathbf{f}_1, \ldots, \mathbf{f}_k$ cost vectors and $\mathbf{b}, \mathbf{d}_1, \ldots, \mathbf{d}_k$ right hand side vectors. We define vectors as column vectors and use the notation $\mathbf{c}^T$ for the transpose of a vector $\mathbf{c}$. We use boldface lowercase letters for vectors and bold uppercase letters for matrices. This problem can be decomposed into a master problem and k different subproblems. The formulation of the subproblems and the resulting Benders cuts differ slightly in the literature. This formulation follows (among others) Bertsimas and Tsitsiklis (1997). The Benders cut is based on the dual objective of the subproblems. The MP is given as:

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{y} + \alpha_1 + \alpha_2 + \cdots + \alpha_k \\ \text{subject to} & \mathbf{A}\mathbf{y} = \mathbf{b} \\ & \textbf{Benders cuts} \\ & \mathbf{y} \in \{0,1\} \end{array}$$

We have for $i = 1, \ldots, k$ the following subproblems where $\bar{\mathbf{y}}$ is the current solution of the master problem.

$$\begin{array}{lrcl} \text{minimize} & \mathbf{f}_i^T\mathbf{x}_i & & \\ \text{subject to} & \mathbf{D}_i\mathbf{x}_i & = & \mathbf{d}_i - \mathbf{B}_i\bar{\mathbf{y}} \\ & \mathbf{x}_i & \geq & 0 \end{array}$$

This subproblem is a linear problem, because the x-variables are continuous. This allows us to construct the dual of the above formulated problem:

$$\begin{array}{ll} \text{maximize} & \mathbf{u}_i^T(\mathbf{d}_i - \mathbf{B}_i\bar{\mathbf{y}}) \\ \text{subject to} & \mathbf{u}_i^T\mathbf{D}_i \leq \mathbf{f}_i^T \\ & \mathbf{u}_i \quad \text{free} \end{array}$$

After solving the subproblems, we update the master problem with additional Benders cuts. These take the following form:

$$\alpha_i \geq \mathbf{p}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y}) \tag{B.1}$$

where $\mathbf{p}_i$ is the vector of dual values corresponding to an optimal solution of subproblem i. Since $\mathbf{p}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y})$ corresponds to the objective of the dual solution, strong duality guarantees that α_i is chosen at least as large as the optimal value of the subproblem when $\mathbf{y} = \bar{\mathbf{y}}$. If the subproblem is infeasible, we add the cut:

$$0 \geq \mathbf{w}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y}) \tag{B.2}$$

where $\mathbf{w}_i$ is an unbounded ray for the dual problem. We will not formally define an unbounded ray because it would also need the notion of other linear algebra concepts. A formal definition could for example be found in Bertsimas and Tsitsiklis (1997). Informally an unbounded ray is a direction in the solution space that will always lead to

more feasible solutions. Multiplying an unbounded ray with a constant will still lead to another feasible solution, no matter how large this constant is. Proceeding via this unbounded ray will lead to an improved objective function which makes the optimization problem unbounded.

Note that the assumption that $\mathbf{f} \geq \mathbf{0}$ is sufficient for $\mathbf{u} = \mathbf{0}$ to be a feasible solution of the dual. Therefore the dual has either an optimal solution equal to the primal optimal solution or the dual has an unbounded ray corresponding to an infeasible primal. If in the master problem $\mathbf{y}$ is chosen such that $\mathbf{w}_i^T(\mathbf{d}_i - \mathbf{B}_i\mathbf{y}) \geq 0$, then the dual of the subproblem becomes unbounded. This means that the primal of the subproblem is infeasible and therefore the complete problem is infeasible. This justifies the use of cut (B.2).

B.1.1 Primal objective Benders cuts

In Castillo et al. (2006) a different kind of subproblem formulation and a different kind of cut is used. They take $\mathbf{y}$ as variable in the subproblem, but add a constraint that fixes $\mathbf{y} = \bar{\mathbf{y}}$. Therefore $\mathbf{y}$ can be treated as a continuous variable instead of a binary variable.

$$\begin{array}{lrl} \text{minimize} & \mathbf{f}_i^T\mathbf{x}_i & \\ \text{subject to} & \mathbf{D}_i\mathbf{x}_i & +\mathbf{B}_i\mathbf{y} = \mathbf{d}_i \\ & \mathbf{y} & = \bar{\mathbf{y}} \\ & \mathbf{x}_i \geq 0; & \mathbf{y} \text{ free} \end{array}$$

Because the y-variables are treated as continuous variables we can construct the dual for this subproblem.

$$\begin{array}{lr} \text{maximize} & \mathbf{u}_i^T\mathbf{d}_i + \mathbf{v}_i^T\bar{\mathbf{y}} \\ \text{subject to} & \mathbf{u}_i^T\mathbf{D}_i \leq \mathbf{f}_i^T \\ & \mathbf{u}_i^T\mathbf{B}_i + \mathbf{v}_i^T = \mathbf{0} \\ & \mathbf{u}_i, \mathbf{v}_i \quad \text{free} \end{array}$$

The Benders cut used in Castillo et al. (2006) is different than the one introduced before. They add the following cut to the master problem, based on the optimal dual variable values $\mathbf{v} = \lambda$ corresponding to the constraints $\mathbf{y} = \bar{\mathbf{y}}$.

$$\alpha_i \geq \mathbf{f}_i^T\bar{\mathbf{x}}_i + \lambda(\mathbf{y} - \bar{\mathbf{y}}) \tag{B.3}$$

In this cut, λ are the optimal dual values corresponding to $\mathbf{y} = \bar{\mathbf{y}}$. By introducing this cut to the master problem, it "learns" that choosing $\mathbf{y} = \bar{\mathbf{y}}$ will lead to objective value $\mathbf{f}_i^T\bar{\mathbf{x}}_i$ for subproblem i. Non-zero dual values correspond to constraints that are active in the subproblem. Adjusting the right hand side for these constraints in the subproblem can lead to smaller optimal values. The dual values can tell us if changing some variables $\mathbf{y}$ might lead to better overall solutions.

Instead of using an unbounded ray of the dual problem when the primal is infeasible, in Castillo et al. (2006) artificial variables are introduced to the primal to create always

feasible subproblems. An artificial variable is added to every constraint. This variable is zero when a constraint is satisfied, but is nonzero otherwise. It takes the value that is needed to satisfy this specific constraint. Minimizing over the artificial variables will lead to "as feasible as possible" solutions. The structure of the Benders cut remains the same, although this adjustment influences the dual values. This leads to the following subproblem formulation (where M is a large enough number):

$$\begin{array}{lll} \text{minimize} & \mathbf{f}_i^T\mathbf{x}_i & +M\cdot\mathbf{1}^T\mathbf{v}+M\cdot w \\ \text{subject to} & \mathbf{D}_i\mathbf{x}_i & +\mathbf{v}-w=\mathbf{d}_i-\mathbf{B}_i\mathbf{y} \\ & \mathbf{y} & =\bar{\mathbf{y}} \\ & \mathbf{x}_i & \geq 0 \end{array}$$

B.2 Classical Benders decomposition for Model1

In this section classical Benders decomposition is implemented for Model1. We choose to use Model1 combined with the additional Constraint (5.9) as a basis, because it further restricts the solution space of the master problem. Every iteration the algorithm generates new cuts for the master problem. These constraints (5.9) can be seen as first cuts to the MP. It is useful since in the MP the sub-tour elimination constraints (5.6) are absent. This drastically increases the feasible region for our x variables. Below we will give the formulations of the master problem and the subproblem.

B.2.1 Master problem

$$MP: \quad \text{minimize} \quad \alpha \tag{B.4}$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1..n \tag{B.5}$$

$$\sum_{l=1}^{m} y_{jl}\cdot l - \sum_{k=1}^{m} y_{ik}\cdot k \geq f(b_i, b_j) \qquad i,j = 1..n \tag{B.6}$$

$$y_{ik} + y_{jk} - 1 \leq x_{ij} + x_{ji} \qquad i,j = 1..n; k = 1..m \tag{B.7}$$

$$x_{ij} + x_{ji} \leq y_{ik} - y_{jk} + 1 \qquad i,j = 1..n; k = 1..m \tag{B.8}$$

$$\alpha - \sum_{i=1}^{n} t_i^{(\kappa)} - M_2 \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} v_{ij}^{(\kappa)} \geq \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} \lambda_{ij}^{(\kappa)}(x_{ij} - x_{ij}^{(\kappa)}) \qquad \kappa = 1..\nu-1 \tag{B.9}$$

$$\alpha \geq 0; \;\; x_{ij}, y_{ik} \in \{0,1\} \qquad i,j = 1..n \tag{B.10}$$

Constraints (B.9) give the added Benders cuts. Note that we use the "primal objective Benders cuts" as described above. The left-hand side contains the artificial variable α

and the objective of the subproblem introduced next. The right-hand side contains the dual values of the optimal solution of the subproblem. In every iteration κ, such a cut is added to the master problem, so in iteration ν there are $\nu - 1$ cuts available. The Benders cut is generated at the end of an iteration. It uses the values of $\mathbf{x}$ found in the master problem and the values of $\lambda, \mathbf{v}$ and $\mathbf{t}$ found in the subproblem. These values will be specified after the subproblem formulations.

B.2.2 Subproblems

Subproblem formulation:

$$\text{SP1:} \quad \text{minimize} \sum_{i=1}^{n} t_i \tag{B.11}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, \ldots, n \tag{B.12}$$

$$t_j - t_i \geq d_{ij} + h_j - (1 - x_{ij}^{(\kappa)})M \qquad i, j = 1..n;\ i \neq j \tag{B.13}$$

For some fixed values $x_{ij}^{(\kappa)}$, the subproblem might become infeasible. This is the case when the master problem assigns subtours in $\mathbf{x}$. To overcome this problem, artificial variables are added to create a feasible subproblem.

$$\text{SP2:} \quad \text{minimize} \sum_{i=1}^{n} t_i + M_2 \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} v_{ij} \tag{B.14}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, \ldots, n \tag{B.15}$$

$$t_j - t_i + v_{ij} \geq d_{ij} + h_j - (1 - x_{ij}^{(\kappa)})M \qquad i, j = 1..n;\ i \neq j \tag{B.16}$$

In this second subproblem, M_2 is a large enough constant. Large enough means in this case that $M_2 \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} v_{ij}$ is at least larger than the optimal objective value. If SP1 is feasible in an iteration we do not solve SP2. Therefore we choose the values of $\mathbf{v}^{(\kappa)}$ zero. The values of λ are the dual variables corresponding to constraints (B.13) and (B.16) multiplied by M. This multiplication with M is needed because the variables x_{ij} are multiplied by M in constraints (B.13) and (B.16). Because of this, we do not need to add continuous variables x_{ij} to the subproblem which shortens the runtime for solving the subproblem. Note that we do not use the dual variables in the subproblems corresponding with constraints (B.12) and (B.15).

B.2.3 Algorithm

Either subproblem1 or subproblem2 returns a n-dimensional vector $t^{(\nu)}$ and a $n(n-1)$-dimensional vector $\lambda^{(\nu)}$. This provides an upper bound UB for the complete problem,

since both the master problem and the subproblem are feasible. The master problem provides us with a master problem feasible configuration of $\mathbf{x}$ and $\mathbf{y}$. It returns a lower bound LB for the optimal value of the complete problem. At first LB is zero, but by adding constraints in each iteration it is pushed towards the optimal value of the complete problem. Since the master problem is restricted in every iteration, LB is a monotonically increasing function for the iteration counter.

Algorithm 10 Classical Benders decomposition Model1

```
while LB<UB do
    solve MP
    x̄ = x
    ȳ = y
    Gain LB
    solve SP1 given x̄
    if infeasible then
        solve SP2 given x̄
    end if
    Gain λ
    Gain UB
    Add Benders cut
end while
```

In an iteration of the algorithm, the master problem is solved first. Solving the master problem returns a lower bound for the optimal value of the complete problem and some values of x and y are found. These values are used for solving subproblem 1. In case subproblem 1 is infeasible, subproblem 2 is solved containing the artificial variables. If subproblem 1 returns an optimal value, this can be used as an upper bound. The variables $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are feasible for the master problem and $\bar{\mathbf{t}}$ is feasible for the subproblem. So $\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$ and $\bar{\mathbf{t}}$ are feasible for the complete problem. In the last step, the benders cut is added to the master problem. This cut teaches the master problem that choosing $\mathbf{x} = \bar{\mathbf{x}}$ and $\mathbf{y} = \bar{\mathbf{y}}$ will lead to an optimal value of $\alpha = UB$. It also tells the master problem which variables to change. If $\lambda_{ij} \neq 0$, then changing the corresponding variable x_{ij} might lead to a better solution for subproblem 1.

B.2.4 Experimental results

The runtime of this algorithm is very long. Therefore it is difficult to measure the performance of this algorithm. The decomposition should work better for larger instances. However, we can only solve problems with 10 jobs in reasonable time. Therefore it is difficult to predict the growth rate as a function of the number of jobs (n) and the number of cranes (m). In Table B.1 the runtime is presented for problems with

$n = 5, m = 2, \theta = 20m$ and $T = 15$ minutes. So we schedule 2.5 jobs per crane in a time interval of 15 minutes. The arrival rate of jobs is thereby equal to the experiments in Chapter 9. The number of bays considered is equal to $20m$ which corresponds to a busy yard lane. In table B.1 Model1 refers to solving Model1 directly. Benders1a refers to the algorithm presented above and Benders1b refers to the algorithm that follows when a dual objective based Benders cut is used.

Number	Delay	Model1	Benders1a	Benders1b
1	0	0.0020	0.0130	0.0090
2	9.0000	0.0020	0.0430	0.0490
3	0	0.0010	0.0030	0.0040
4	110.0000	0.0050	0.1080	0.1240
5	0	0.0010	0.0310	0.0290
6	9.0000	0.0040	0.0220	0.0260
7	15.0000	0.0030	0.0290	0.0260
8	2.0000	0.0030	0.0120	0.0160
9	17.0000	0.0010	0.0410	0.0440
10	90.0000	0.0070	0.2580	0.2570
11	28.0000	0.0030	0.0420	0.0610
12	25.0000	0.0040	0.0430	0.0380
13	7.0000	0.0020	0.0040	0.0040
14	37.0000	0.0030	0.0650	0.0520
15	6.0000	0.0020	0.0120	0.0140
16	35.0000	0.0020	0.0190	0.0100
17	0	0.0010	0.0250	0.0040
18	41.0000	0.0020	0.0740	0.0470
19	23.0000	0.0010	0.0210	0.0340
20	0	0.0020	0.0010	0.0020
avg	22.7000	0.0025	0.0433	0.0425

Table B.1: Classical Benders: $n = 5, m = 2$ and $T = 15$ minutes

The runtime of the algorithms is within a second for this extremely small problems. The decomposition methods take more time than solving directly, but that is not surprising. Decomposition methods are designed for large instances, not for small instances. There seems to be little difference in runtime between the different Benders cuts. If we increase the problem size to $n = 10$ and $m = 2$ we already have much longer runtime. Especially the runtime of the Benders decomposition algorithms increases drastically. For this case we runned only 10 problem instances because the runtime was much larger and the results were very clear.

The long runtime of the decomposition algorithms is due to a huge number of iterations (more than 1000 iterations for $n = 10$). The Benders cuts fail to exclude many

Number	Delay	Model1	Benders1a	Benders1b
1	44	0.0070	40.3563	21.4902
2	109	0.0170	54.5481	26.2985
3	209	0.0220	26.9276	20.4672
4	6	0.0040	25.1044	7.9334
5	50	0.0080	7.5404	20.8182
6	38	0.0060	37.6891	21.5702
7	48	0.0080	24.9604	19.0841
8	96	0.0120	81.0167	39.8023
9	6	0.0040	22.6463	18.7541
10	55	0.0100	41.6173	35.2590
Avg	66.1	0.0098	36.2407	23.1477

Table B.2: Classical Benders: $n = 10, m = 2$ and $T = 30$ minutes

solutions for the master problem. Therefore many cuts are needed to find an optimal solution. This required number of iterations seems to grow exponentially with respect to the problem size. The difference in runtime between the different decomposition algorithms seems to come from the amount of time it takes to solve the master problem. The primal objective based Benders cuts are apparently more difficult than the dual objective based Benders cut. For both algorithms the number of iterations needed is similar, but the time per iteration grows for the primal objective based decomposition. For the dual objective based Benders decomposition method the time it takes Gurobi to solve the master problem grows less than for the primal objective based Benders decomposition.

B.3 Classical Benders decomposition for Model3

Benders decomposition is proposed for the mTSP problem by Bektaş (2012). It might therefore also gain significant runtime reduction for our Model3 since it shows a lot of similarities. In Bektaş (2012) the best performance was found when the subtour elimination constraints were in the subproblems, so we will do the same for this decomposition.

B.3.1 Problem formulations

As is done in Bektaş (2012) we will add the constraints $x_{ij} + x_{ji} \leq 1$ to the master problem to eliminate subtours of size 2. This results in two new subproblems, one with constraints (5.27), (5.28) and (5.29) and one with constraints (5.30) and (5.31). This will give us the following problem definitions:

Master problem

$$\text{minimize} \quad \alpha \tag{B.17}$$

$$\sum_{j=1}^{m+n} x_{hj} = 1 \qquad h = 1, \ldots, m+n \tag{B.18}$$

$$\sum_{i=1}^{m+n} x_{ih} = 1 \qquad h = 1, \ldots, m+n \tag{B.19}$$

$$x_{ij} + x_{ji} \leq 1 \qquad i, j = m+1, \ldots, m+n \tag{B.20}$$

$$\text{Benders Cuts} \tag{B.21}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, m+n \tag{B.22}$$

The Benders cuts are defined later, because they depend on the solution of the subproblems.

Subproblem1

$$\text{minimize} \quad 0 \tag{B.23}$$

$$k_d = d \qquad d = 1, \ldots, m \tag{B.24}$$

$$k_i - k_j \leq m - 1 - (m-1)\bar{x_{ij}} \qquad i, j = 1, \ldots, m+n \tag{B.25}$$

$$k_j - k_i \geq f(b_i, b_j) \qquad i, j = m+1, \ldots, m+n \tag{B.26}$$

Subproblem2

$$\text{minimize} \quad \sum_{i \in \mathcal{C}} t_i \tag{B.27}$$

$$t_i \geq r_i + h_i \qquad i, j = 1, \ldots, m+n \tag{B.28}$$

$$t_j - t_i \geq d_{ij} + h_j - M + M\bar{x_{ij}} \qquad i = m+1, \ldots, m+n \tag{B.29}$$

For this second model the dual objective Benders cuts are used. This method is chosen because we encounter many infeasible subproblems while solving via Benders decomposition. The dual objective Benders cuts give us the opportunity to directly use an unbounded ray for a Benders cut. In addition, it gives the opportunity to demonstrate this dual objective method. Note that both subproblems are LP, so they can be efficiently solved.

B.3.2 Algorithm

After solving the master problem we first check if this solution is feasible for subproblem 1. If this is not the case, a Benders cut can be constructed directly. When subproblem 1 is feasible, we check whether subproblem 2 is feasible. Depending on this outcome we generate two different Benders cuts. Note that we cannot construct a good Benders cut based on subproblem 1 when it is feasible. In the master problem as wel as subproblem 1 an objective is absent. An objective is needed to construct Benders cuts. We use algorithm 11 to solve the problem (BC1, BC2 and BC3 will be specified later).

Algorithm 11 Classical Benders decomposition Model3

```
while LB<UB do
    solve MP
    x̄ = x
    Update LB
    solve SP1 given x̄
    if infeasible then
        Add BC1 to MP
    else
        solve SP2 given x̄
        if infeasible then
            Add BC2 to MP
        else
            Add BC3 to MP
            Update UB
        end if
    end if
end while
```

This algorithm will break down when an optimal solution is found ($LB = UB$). When both subproblems are feasible, we have a feasible solution to the complete problem and therefore an upper bound can be calculated. BC3 will ensures that the LB increases during the algorithm. The current solution remains feasible for the MP with optimal value $\alpha = UB$. Since α is minimized, the MP will never find an optimal $\mathbf{x}$ with $\alpha > UB$. When we add a Benders cut to the Master Problem, we reduce its feasible set. BC1 and BC2 ensure that at least the current solution is excluded, while BC3 either corresponds to an optimal solution or it excludes other solutions. Since the feasible set for the Master problem is finite in X, the algorithm will terminate in a finit number of iterations.

B.3.3 Benders cuts

We will give the definition of the Benders cuts. BC1 involves an unbounded ray w_1 for the dual problem of subproblem1 since subproblem1 is infeasible. If subproblem2 is infeasible, we get an unbounded ray w_2 for the dual problem. Existence of such an unbounded ray follows for both problems from a non-negative objective vector and non-negative variables. If subproblem2 is feasible, we get the optimal dual vector p_2.

$$w_1^T(d_1 - B_1x) \leq 0 \qquad \text{(BC1)}$$
$$w_2^T(d_2 - B_2x) \leq 0 \qquad \text{(BC2)}$$
$$p_2^T(d_2 - B_2x) \leq \alpha \qquad \text{(BC3)}$$

where

$$d_1 = [1, 2, \ldots, m, m-1, m-1, \ldots, m-1, \frac{m-1}{2}\big(\text{sign}(x_i - x_j + \epsilon) - 1\big), \ldots]^T$$

$$d_2 = [r_1 + h_1, \ldots, r_n + h_n, d_{ij} + h_j - M, \ldots]^T$$

and

$$B_1 = \begin{Bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ m-1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & m-1 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{Bmatrix}, \quad B_2 = \begin{Bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ -M & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & -M \end{Bmatrix}.$$

B.3.4 Experimental results

The computation times for solving Model3 directly was quite long. As for Model1 the classical Benders decomposition method did not speed up the calculations for small problem instances. Since we want to solve instances within two minutes, algorithm 11 runs out of time for instances of size $n \geq 10$. To illustrate the long runtime and the growth of this runtime as function of n, we present some results for $n = 5$ in table B.3. The average time it took for algorithm 11 to solve problems of size $n = 5$ was 1.44 seconds. For $n = 10$ it usually runs out of time.

This decomposition decomposes the problem in two different subproblems in contrast to Model1. This can be an advantage, because two cuts can be introduced per iteration. On the contrary, there is less information in the master problem. Besides the sub tour

Number	Delay	Model1	Model3	Model3 Benders
1	0	0.0020	0.0040	0.1030
2	0	0.0010	0.0050	0.1410
3	42	0.0030	0.0070	3.0732
4	7	0.0020	0.0060	1.9131
5	43	0.0020	0.0090	3.4312
6	12	0.0020	0.0050	0.7250
7	13	0.0040	0.0090	1.2061
8	17	0.0010	0.0050	0.3570
9	51	0.0040	0.0140	2.7192
10	40	0.0030	0.0100	0.7730
Avg	22.5	0.0024	0.0074	1.4442

Table B.3: Standard Benders for Model3: $n = 5, m = 2$ and $T = 15$ minutes

elimination constraints, also the non-crossing constraints are in a subproblem. Therefore the master problem finds often solutions that lead to infeasible subproblems. This results in many iterations. It is found that the first subproblem is infeasible in about half of the instances. If the first subproblem is feasible, on average one third of the times subproblem 2 is infeasible. This decomposition algorithm finds in about one third of the iterations a feasible solution that provides an upper bound.

B.4 Combinatorial Benders cuts for Model1

As suggested by Codato and Fischetti (2006) a combinatorial Benders cut (CB cut) might work well for the "big-M" constraint. Model1 uses such a "big-M" constraint, so it seems applicable for a combinatorial Benders decomposition. The same idea as for the classical Benders decomposition holds, only the cut introduced to the master problem is different. Instead of using the dual variables, it just forms a minimal combination of conflicting constraints for infeasible subproblems. Since our objective is in the subproblem, the master problem just has to search for feasible solutions. If in an iteration the subproblem is infeasible, a procedure "MISsearch" is used to find a minimal infeasible subsystem of the subproblem. This procedure finds constraints that are conflicting for the subproblem such that as few constraints are used as possible. Since constraints of type (B.40) correspond to x variables in the master problem, using as few constraints as possible result in using few variables in the Benders cut. The advantage of less variables in the cut, is that more solutions are cut out. The cut is stronger and therefore less iterations will be needed. The variables x that are concerned with these constraints are used to construct the cut. This CB cut simply prevents the master problem from choosing these values for x. Let $C = \{i, j \mid$ The constraint of type (B.40) containing x_{ij} is in the MIS$\}$

$$\sum_{i,j \in C : \bar{x}_{ij}=0} x_{ij} + \sum_{i,j \in C ; \bar{x}_{ij}=1} (1 - x_{ij}) \geq 1 \tag{B.30}$$

This results in the following problem formulations:

Master problem:

$$\text{minimize} \quad 0 \tag{B.31}$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1..n \tag{B.32}$$

$$\sum_{l=1}^{m} y_{jl} \cdot l - \sum_{k=1}^{m} y_{ik} \cdot k \geq f(b_i, b_j) \qquad i, j = 1..n \tag{B.33}$$

$$y_{ik} + y_{jk} - 1 \leq x_{ij} + x_{ji} \qquad i, j = 1..n; k = 1..m \tag{B.34}$$

$$x_{ij} + x_{ji} \leq y_{ik} - y_{jk} + 1 \qquad i, j = 1..n; k = 1..m \tag{B.35}$$

$$\sum_{\substack{(i,j) \in C^{\kappa} \\ x_{ij}=0}} x_{ij} + \sum_{\substack{(i,j) \in C^{\kappa} \\ x_{ij}=1}} (1 - x_{ij}) \geq 1 \qquad \kappa = 1, .., \nu - 1 \tag{B.36}$$

$$x_{ij}, y_{ik} \in \{0, 1\} \qquad i, j = 1..n \tag{B.37}$$

Subproblem:

$$\text{minimize} \quad \sum_{i=1}^{n} t_i \tag{B.38}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, \ldots, n \tag{B.39}$$

$$t_j - t_i \geq d_{ij} + h_j - (1 - x_{ij}^{(\nu)})M \qquad i, j = 1..n;\ i \neq j \tag{B.40}$$

$$\sum_{i=1}^{n} t_i \leq UB - \epsilon \tag{B.41}$$

$$t_j \geq 0 \qquad i = 1..n \tag{B.42}$$

B.4.1 MIS search

Since a CB cut is generated in every iteration, the feasible solution space of the master problem decreases in every iteration. At least one combination of x values is excluded. Note that it is usefull to find a small infeasible set of constraints because if the set has few elements, a large number of possible combinations are excluded. To find a infeasible set of constraints, we use the "MISsearch" procedure as suggested by Codato and Fischetti (2006). In the subproblem, we set the objective function to zero. Next we construct the dual.

Subproblem*

$$\text{minimize} \quad 0 \tag{B.43}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, \ldots, n \tag{B.44}$$

$$t_j - t_i \geq d_{ij} + h_j - (1 - \mathbf{x}_{ij}^{(\nu)})M \qquad i, j = 1..n;\ i \neq j \tag{B.45}$$

$$\sum_{i=1}^{n} t_i \leq UB - \epsilon \tag{B.46}$$

Dual*

$$\text{maximize} \quad \sum_{i=1}^{n} \mu_i (r_i + h_i) + \sum_{i,j=1}^{n} \lambda_{ij} \left(d_{ij} + h_j - (1 - \mathbf{x}_{ij}^{(\nu)})M \right) + \rho(UB - \epsilon) \tag{B.47}$$

$$\sum_{j=1}^{n} \mu_j + \sum_{j=1, j\neq i}^{n} \lambda_{ij} + \sum_{j=1, j\neq i}^{n} \lambda_{ji} + \rho = 0 \quad i = 1, 2, \ldots, n \tag{B.48}$$

$$\mu_i \geq 0 \tag{B.49}$$

$$\lambda_{ij} \geq 0 \tag{B.50}$$

$$\rho \leq 0 \tag{B.51}$$

Since the subproblem is infeasible, the dual must be unbounded or also infeasible. Since $\mu = 0$, $\lambda = 0$ and $\rho = 0$ is a feasible solution to the dual problem, this means that the dual is unbounded. If the dual is unbounded, this must mean that there is an unbounded ray and that the dual objective can take any value by multiplying this unbounded ray by a constant. In particular, there exists an feasible solution with objective equal to 1. It is therefore guarantied that the following "extended dual" is feasible.

Dual

$$\text{maximize} \quad 0 \tag{B.52}$$

$$\sum_{i=1}^{n} \mu_i(r_i + h_i) + \sum_{i,j=1}^{n} \lambda_i j \left(d_{ij} + h_j - (1 - \mathbf{x}_{ij}^{(\nu)})M\right) + \rho(UB - \epsilon) = 1 \tag{B.53}$$

$$\sum_{j=1}^{n} \mu_j + \sum_{j=1, j\neq i}^{n} \lambda_{ij} + \sum_{j=1, j\neq i}^{n} \lambda_{ji} + \rho = 0 \quad i = 1, 2, \ldots, n \tag{B.54}$$

$$\mu_i \geq 0 \tag{B.55}$$

$$\lambda_{ij} \geq 0 \tag{B.56}$$

$$\rho \leq 0 \tag{B.57}$$

Now we can introduce an objective function for the dual problem to find dual variables with the desired characteristics. We are interested in finding a minimal set of positive dual values for the linking constraints. So we minimize over λ and introduce the objective function: $\min \sum_{i,j=1, i\neq j}^{n} \lambda_{ij}$. Note that MIS search is a heuristic procedure and is not guaranteed to find a MIS, but always finds an infeasible subset. If this infeasible subset of constraints contains a constraint of type (B.40), the master problem is restricted. If no constraint of type (B.40) is in the infeasible subset, an optimal solution has been found. It means that only the constraints of type (B.39) fail to find a solution better than the current upper bound. Therefore, a solution with zero delay can be found which is always optimal. Because MIS search finds an optimal solution or introduces a cut restricting the master problem, the algorithm converges to an optimal solution.

B.4.2 Algorithm

We use Algorithm 12 to solve Model1 using combinatorial Benders decomposition. This Algorithm breaks down when the master problem becomes infeasible. This means that there are no values of x, y left that might lead to better solution. The best solution found thus far is the optimal solution. Since the solution space of the master problem is reduced in every iteration and the solution space of the master problem is finite, the algorithm is guarantied to terminate. The CB cuts only cuts out solutions with a worse or equal objective value than the current best. Therefore optimality is guarantied.

Algorithm 12 Combinatorial Benders decomposition Model1

```
UB= ∞
while do
    solve MP
    if infeasible then
        break
    end if
    x̄ = x
    ȳ = y
    solve SP given x̄
    if feasible then
        update UB and solve SP again
    end if
    Solve dual
    uptain λ
    Create new CB cut for the MP
    Add CB cut to the MP
end while
```

B.4.3 Experimental results

Number	Delay	Model1	Combinatorial Benders
1	93	0.0160	91.2492
2	94	0.0170	11.3216
3	60	0.0110	6.6544
4	90	0.0170	1.1671
5	145	0.0190	13.6938
6	91	0.0170	10.5246
7	62	0.0080	2.1621
8	31	0.0050	6.0644
9	29	0.0060	4.3302
10	38	0.0070	2.9272
avg	73.3	0.0123	15.0095

Table B.4: Combinatorial Benders: $n = 10, m = 2$ and $T = 30$ minutes

The runtime of this decomposition method is better than for the classical Benders decomposition. Compared to the classical decomposition method this method finds the optimal solution faster due to less iterations. However, the runtime is much longer than direct solving. The number of iterations is still too high to gain better performance. The master problem is solved faster than directly solving the complete problem, but the number of iterations needed increases with the problem size. In Table B.4 we show the

results of problems with size $n = 10$, similar to the experiments described in subsection B.2. If the problem size increases, the time to solve the master problem becomes small compared to solving directly. For $n = 20$, initially the master problem is solved a factor 7 faster than solving the complete problem. This factor is likely to increase for larger instances. However, the number of iterations needed increases. The reduced runtime for the master problem seems not significant compared to the growth in the number of iterations.

The master problem is not directed to prefer $x_{ij} = 0$ above $x_{ij} = 1$. If both are feasible it can very well choose $x_{ij} = 1$. This will never lead to a better solution, so must be discouraged. This can be done by adding an objective to the master problem that minimizes the sum of all x_{ij}. However, this resulted in much longer runtime for the master problem since it has to optimize above finding just a feasible solution. Often the master problem takes longer to solve once than solving the complete problem directly. Runtime might also be reduced by introducing multiple cuts in each iteration. This is suggested in Codato and Fischetti (2006) but research must be done to decide how many cuts should be introduced. A smart way needs to be developed to choose cuts that exclude different "bad" solutions for the master problem without increasing the runtime for the master problem by too much.

B.5 Double Benders decomposition

In the previous examples, we tried to split the problem between binary and continuous variables. It is also possible to split the problem between the y and the x variables. This creates an easier master problem, but more difficult subproblems. In Chu and Xia (2004) the authors introduce an integer Benders cut. This cut can be used to decompose in two integer problems. Because we cannot have continuous variables in our subproblem, we need to decompose twice. The integer Benders cuts (CGPr-cuts) are explained later. The problem in y becomes the master problem (MP) and becomes an assignment problem. The problem in x is the intermediate problem (IP) and the problem in t becomes the subproblem (SP). We start with finding a feasible instance for the y variables. Then, the intermediate problem in x is solved to optimality (given y). To solve for x we solve the continuous problem for t given specific values of x. Repeatedly solving for t gives us an optimal x given y. This results in a new cut for the master problem. While the master problem remains feasible (new feasible instances of y can be found) this procedure is repeated. Once the master problem becomes infeasible we can stop and the last solution found is the optimal one.
While solving an intermediate problem in x, we add CB-cuts to our intermediate problem. These cuts are independent of y and therefore valid for all instances of y. The advantage of this, is that we can keep the CB-cuts in our intermediate problem for all iterations to come. When we arrive in the intermediate problem again, some cuts are already available which might speed up the process. Violating these cuts has already

proven to generate worse or infeasible solutions. Experimental results show that indeed the number of iterations in the x, t loop decreases for subsequent y, x iterations.

Master problem (MP)

$$\text{minimize} \quad 0 \tag{B.58}$$

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i = 1..n \tag{B.59}$$

$$\sum_{l=1}^{m} y_{jl} \cdot l - \sum_{k=1}^{m} y_{ik} \cdot k \geq f(b_i, b_j) \qquad i, j = 1..n \tag{B.60}$$

$$CGPR - cuts \tag{B.61}$$

$$x_{ij}, y_{ik} \in \{0, 1\} \qquad i, j = 1..n \tag{B.62}$$

Intermediate problem (IP)

$$\text{minimize} \quad 0 \tag{B.63}$$

$$x_{ij} + x_{ji} \geq \bar{y}_{ik} + \bar{y}_{jk} - 1 \qquad i, j = 1..n; k = 1..m \tag{B.64}$$

$$x_{ij} + x_{ji} \leq \bar{y}_{ik} - \bar{y}_{jk} + 1 \qquad i, j = 1..n; k = 1..m \tag{B.65}$$

$$\sum_{\substack{(i,j) \in C^{\kappa} \\ x_{ij}=0}} x_{ij} + \sum_{\substack{(i,j) \in C^{\kappa} \\ x_{ij}=1}} (1 - x_{ij}) \geq 1 \qquad \kappa = 1, .., \nu - 1 \tag{B.66}$$

$$x_{ij}, y_{ik} \in \{0, 1\} \qquad i, j = 1..n \tag{B.67}$$

Subproblem (SP)

$$\text{minimize} \quad \sum_{i=1}^{n} t_i \tag{B.68}$$

$$t_i \geq r_i + h_i \qquad i = 1, 2, \ldots, n \tag{B.69}$$

$$t_j - t_i \geq d_{ij} + h_j - (1 - x_{ij}^{(\nu)})M \qquad i, j = 1..n;\ i \neq j \tag{B.70}$$

$$\sum_{i=1}^{n} t_i \leq UB - \epsilon \tag{B.71}$$

$$t_j \geq 0 \qquad i = 1..n \tag{B.72}$$

B.5.1 Algorithm

Algorithm 13 is used to solve Model1 using this double Benders decomposition. It follows the structure that is visualized in Figure B.1. The CB cuts are similar to the cuts presented in the previous subsection. The CGP-cuts are based on Chu and Xia (2004) which are presented next.

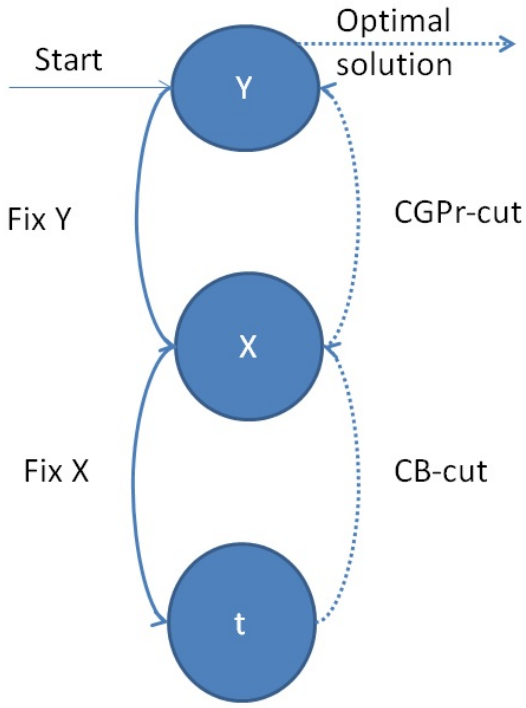


Figure B.1: Double Benders

B.5.2 CGP-cuts

When the intermediate problem given y is solved to optimality, we need to introduce a cut for the master problem. This is done via a CGPR-cut, introduced in Chu and Xia (2004). This cut generations method is quit technical and involves solving an MILP. A detailed description of the procedure can be found in Chu and Xia (2004), so only the general idea is sketched here. Let us rewrite our intermediate problem to a generic vector formulation where A, B and b are chosen such that they correspond to our intermediate problem in a certain iteration (some constraints have to be multiplied by -1 to obtain $\leq$ in the formulation). Note that this formulation includes the generated combinatorial Benders cuts by the subproblem. Furthermore, we introduce the notation IP($\bar{\mathbf{y}}$) to show the dependence of our intermediate problem on $\bar{\mathbf{y}}$.

$$
\begin{array}{lrcl}
\text{IP'}(\bar{\mathbf{y}}) & & & \\
\text{minimize} & 0 & & \\
\text{subject to} & \mathbf{Bx} & \leq & \mathbf{b} - \mathbf{A}\bar{\mathbf{y}}
\end{array}
$$

$$
\mathbf{x} \in \{0,1\}^{n(n-1)}
$$

Algorithm 13 Double Benders decomposition

```
while MP feasible do
    Solve MP
    Update IP with y
    while IP feasible do
        Solve IP
        Update SP
        Solve SP
        if feasible then
            update UB and solve SP again
        end if
        Solve dual
        Obtain λ
        Create new CB cut for the IP
        Add CB cut to the IP
    end while
    Create CGPr-cut for MP
    Add CGPr-cut for MP
end while
```

Note that this problem is infeasible due to the added combinatorial Benders cuts. We therefore introduce artificial variables to make it feasible. We minimize over these artificial variables and therefore these artificial variables will represent "the amount of violation" for each constraint. This gives us the following always feasible intermediate problem.

$$\begin{array}{lrcl} \text{IP}(\bar{\mathbf{y}}) & & & \\ \text{minimize} & \mathbf{1}^T\mathbf{r} & & \\ \text{subject to} & \mathbf{Bx} - \mathbf{r} & \leq & \mathbf{b} - \mathbf{A}\bar{\mathbf{y}} \end{array}$$

$$\mathbf{x} \in \{0,1\}^{n(n-1)}, \mathbf{r} \geq 0$$

Because $\text{IP}(\bar{\mathbf{y}})$ is an integer program we cannot directly construct a dual problem. We therefore look at a fixed version of $\text{IP}(\bar{\mathbf{y}})$. We fix the values of $\mathbf{x}$ to $\bar{\mathbf{x}}$ and gain $2^{n(n-1)}$ (there are $n(n-1)$ x variables) different fixed subproblems:

$$\begin{array}{lrcl} \text{IP}(\bar{\mathbf{y}},\bar{\mathbf{x}}) & & & \\ \text{minimize} & 0 & & \\ \text{subject to} & \mathbf{B}\bar{\mathbf{x}} & \leq & \mathbf{b} - \mathbf{A}\bar{\mathbf{y}} \\ & \mathbf{x} & = & \bar{\mathbf{x}} \end{array}$$

$$\mathbf{x} : \text{free}, \mathbf{r} \geq 0$$

For these fixed subproblems, we can construct a dual problem:

$$\begin{aligned} &\text{DIP}(\bar{\mathbf{y}},\bar{\mathbf{x}}) \\ &\text{maximize} \quad (\mathbf{A}\bar{\mathbf{y}} - \mathbf{b})^T \mathbf{u} + \bar{\mathbf{x}}^T \mathbf{v} \\ &\text{subject to} \quad -\mathbf{B}^T \mathbf{u} + \mathbf{v} = 0 \\ &\qquad\qquad\qquad\quad \mathbf{u} \leq 1 \\ &\qquad\qquad\qquad\quad \mathbf{u} \geq 0 \quad \mathbf{v} : \text{free} \end{aligned}$$

Next, a set of constraints is given for an optimal solution to a problem IP($\bar{\mathbf{y}}$,$\bar{\mathbf{x}}$). For an optimal solution $(\bar{\mathbf{y}}^*, \bar{\mathbf{x}}^*)$ the primal, dual and complementary slackness constraints must be satisfied:

$$\mathbf{B}\bar{\mathbf{x}} \leq \mathbf{b} - \mathbf{A}\bar{\mathbf{y}} \tag{B.73}$$

$$\mathbf{x} = \bar{\mathbf{x}} \tag{B.74}$$

$$-\mathbf{B}^T \mathbf{u} + \mathbf{v} = 0 \tag{B.75}$$

$$\mathbf{u} \leq 1 \tag{B.76}$$

$$\mathbf{u}_i \left(-\mathbf{B}\mathbf{x} - \mathbf{A}\bar{\mathbf{y}} + \mathbf{b}\right)_i = 0 \qquad \forall i \tag{B.77}$$

$$\mathbf{r}_i \left(\mathbf{u} - \mathbf{1}\right)_i = 0 \qquad \forall i \tag{B.78}$$

Let $\bar{\mathbf{u}}$ and $\bar{\mathbf{v}}$ be the optimal solution for $\mathbf{u}$ and $\mathbf{v}$ corresponding to $\bar{\mathbf{x}}$. In Chu and Xia (2004) it is shown in Theorem 1 that $\bar{\mathbf{x}}^T \bar{\mathbf{v}} \leq \mathbf{x}^T \bar{\mathbf{v}}$ for all feasible $\mathbf{x}$ is a sufficient condition for $\left(\mathbf{A}\bar{\mathbf{y}} - \mathbf{b}\right)^T \bar{\mathbf{u}} + \bar{\mathbf{x}}^T \bar{\mathbf{v}} \leq 0$ to be a valid integer Benders cut. This sufficient condition is shown to be equivalent with:

$$\bar{\mathbf{x}}_i \bar{\mathbf{v}}_i \leq 0 \qquad \forall i \tag{B.79}$$

$$\left(\mathbf{1} - \bar{\mathbf{x}}\right)_i \bar{\mathbf{v}}_i \geq 0 \qquad \forall i \tag{B.80}$$

The adventage of this formulation is absense of $\mathbf{x}$ in these constraints. We can define a new problem CGP($\bar{\mathbf{y}}$) to find a valid integer Benders cut, where we combine the above constraints. The set of constraints are simplified using $\mathbf{x} = \bar{\mathbf{x}}$ and $\bar{\mathbf{v}} = \mathbf{B}^T \bar{\mathbf{u}}$. Furthermore, constraint (B.78) is shown to be redundant in the article.

$$\text{CGP}(\bar{\mathbf{y}}) \tag{B.81}$$

$$\text{minimize } 0 \tag{B.82}$$

$$\mathbf{B}\bar{\mathbf{x}} \leq \mathbf{b} - \mathbf{A}\bar{\mathbf{y}} \tag{B.83}$$

$$\mathbf{u}_i \left(-\mathbf{B}\mathbf{x}\right)_i = \mathbf{u}_i \left(\mathbf{A}\bar{\mathbf{y}} + \mathbf{b}\right)_i \qquad \forall i \tag{B.84}$$

$$\bar{\mathbf{x}}_i \left(\mathbf{B}^T \mathbf{u}\right)_i \leq 0 \qquad \forall i \tag{B.85}$$

$$\left(\mathbf{1} - \bar{\mathbf{x}}\right)_i \left(\mathbf{B}^T \mathbf{u}\right)_i \geq 0 \qquad \forall i \tag{B.86}$$

$$\bar{\mathbf{x}} \in \{0,1\}^{n(n-1)}, \; \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \tag{B.87}$$

Note that $\bar{\mathbf{x}}$ is a variable in this problem. The aim of this problem is to find a $\bar{\mathbf{x}}$ that generates a valid integer Benders cut.This formulation of CGP is not a linear program, but this can be fixed by introducing more continuous variables. Define $\mathbf{w}_{ij} = \bar{\mathbf{x}}_i \mathbf{u}_j$ by the following constraints:

$$w_{ij} \leq \bar{x}_i \qquad \forall i, j \tag{B.88}$$
$$w_{ij} \leq u_j \qquad \forall i, j \tag{B.89}$$
$$w_{ij} \geq \bar{x}_i + u_j \qquad \forall i, j \tag{B.90}$$

B.5.3 Relaxed CGPr-cut

The above problem is not always feasible due to constraints (B.79) and (B.80). A possible cut $(\mathbf{A}\bar{\mathbf{y}} - \mathbf{b})^T \bar{\mathbf{u}} + \bar{\mathbf{x}}^T \bar{\mathbf{v}} \leq 0$ can be relaxed by inverting some values of $\bar{\mathbf{x}}$ that violate the sign constraints (B.79) and (B.80). Define

$$\bar{\mathbf{x}}'_i = \begin{cases} \bar{\mathbf{x}}_i & \text{if (B.79) and (B.80) are satisfied for the } i\text{th element,} \\ 1 - \bar{\mathbf{x}}_i & \text{otherwise.} \end{cases}$$

In this case $(\mathbf{A}\bar{\mathbf{y}} - \mathbf{b})^T \bar{\mathbf{u}} + (\bar{\mathbf{x}}')^T \bar{\mathbf{v}} \leq 0$ is an integer Benders cut that does satisfy the sign constraints. By this alternative cut a relaxation gap exists and is defined by $\sum_{i=1}^{n} (\bar{\mathbf{x}} - \bar{\mathbf{x}}')_i \bar{\mathbf{v}}_i$. We want this gap to be as small as possible and therefore we use the CGP problem to find a cut which is minimally relaxed. We therefore introduce the problem CGPr which is a relaxed version of CGP and finds the cut for which the relaxation gap is minimal. Introducing also the constraint to make CGP a linear problem, we get the following definition for CGPr which we will use to find CGPr cuts for the MP:

$$\text{CGPr}(\bar{\mathbf{y}}) \tag{B.91}$$
$$\text{minimize} \quad \mathbf{1}^T \mathbf{p} + \mathbf{1}^T \mathbf{q} \tag{B.92}$$
$$\mathbf{B}\bar{\mathbf{x}} \leq \mathbf{b} - \mathbf{A}\bar{\mathbf{y}} \tag{B.93}$$
$$-\sum_j \mathbf{B}_{ij} \mathbf{w}_{ji} = \mathbf{u}_i \left(\mathbf{A}\bar{\mathbf{y}} + \mathbf{b}\right)_i \qquad \forall i \tag{B.94}$$
$$\sum_j \mathbf{B}_{ji} \mathbf{w}_{ij} - \mathbf{p}_i \leq 0 \qquad \forall i \tag{B.95}$$
$$\left(\mathbf{B}^T \mathbf{u}\right)_i - \sum_j \mathbf{B}_{ji} \mathbf{w}_{ij} + \mathbf{q}_i \geq 0 \qquad \forall i \tag{B.96}$$
$$\mathbf{w}_{ij} \leq \bar{\mathbf{x}}_i \qquad \forall i, j \tag{B.97}$$
$$\mathbf{w}_{ij} \leq \mathbf{u}_j \qquad \forall i, j \tag{B.98}$$
$$\mathbf{w}_{ij} \geq \bar{\mathbf{x}}_i + \mathbf{u}_j \qquad \forall i, j \tag{B.99}$$
$$\bar{\mathbf{x}} \in \{0,1\}^{(n(n-1)}, \ \mathbf{0} \leq \mathbf{u} \leq \mathbf{1}, \ \mathbf{p} \geq \mathbf{0}, \ \mathbf{q} \geq 0 \tag{B.100}$$

When this problem is solved, we have found a minimally relaxed integer Benders cut for the Master problem. The solution $\bar{\mathbf{x}}$ can be used to define $\bar{\mathbf{x}}'$ using $\bar{\mathbf{v}} = \mathbf{B}^T\bar{\mathbf{u}}$. This will result in the CGPr-cut:

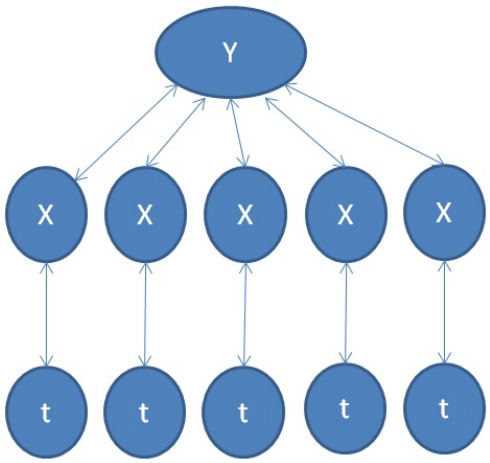
$$(\mathbf{A}\bar{\mathbf{y}} - \mathbf{b})^T\,\bar{\mathbf{u}} + (\bar{\mathbf{x}}')^T\bar{\mathbf{v}} \leq 0 \tag{B.101}$$

If the relaxation of this cut equals zero, we have a valid Benders cut that will cut out the current feasible solution $\bar{\mathbf{y}}$. If this is not the case, (B.101) might or might not cut out the current feasible solution $\bar{\mathbf{y}}$. If it does not cut out $\bar{\mathbf{y}}$ additionally a no-good cut (B.102) is added to the master problem to guaranty convergence.

$$\sum_{k=1}^{m}\sum_{i=1}^{n}\bar{y}_{ik}(1 - y_{ik}) + \sum_{k=1}^{m}\sum_{i=1}^{n}y_{ik}(1 - \bar{y}_{ik}) \geq 1 \tag{B.102}$$

B.5.4 Experimental results

It was not possible to test this double Benders decomposition on large problem instances. This is mainly due to the CGPr sub-routine. This sub-routine takes a lot of time and memory to find good cuts. For problem sizes of $n = 10$, CGPr already runs out of memory. It is therefore not useful for our problem. This cut generation makes it possible to decompose the problem twice, but has only theoretical value to this problem. It might be preferable to not split the problem between x and t. However, the cuts as presented above do not take an objective for the intermediate problem into account. Therefore we cannot directly use the CGPr cuts for a decomposition with x and t in the subproblem. It would be better to decompose the Master problem into different subproblems. This is visualized in figure B.2. In chapter 6 the logic-based Benders decomposition method is explained which worked well for our problem. These cuts can be defined by subproblems containing both x and t. Therefore we do not need the double Benders decomposition. The decomposition between the x variables and the continuous t variables has showed to result in longer runtime.



-Generate a subproblem (X) for every crane

-Generates multiple cuts for (Y) in every iteration, but cuts for (X) cannot be remembered

Figure B.2: Double Benders split problems

Appendix C

Lower bound procedures for Benders decomposition

In Chapter 6 we presented different sublb procedures (lower bound procedures for the single crane subproblem) that could be used as a lower bound for a single crane subproblem. Sometimes unrealistic subproblems are constructed by the master problem that not only will lead to very bad solution, but also very long computation times for the subproblems. These sublb procedures are fast procedures that find a lower bound fast that can be used instead of solving the subproblem. We presented three different sublb procedures that we used in the decomposition algorithm. In this chapter we present the results of testing these procedures. It is important that those sublb procedures find a strong lower bound (a lower bound close to the actual optimal solution) and have short computation time. We also present a fourth sublb procedure as is introduced in Ng and Mak (2005). In the first section this sublb procedure is introduced. In the next section we compare the four different sublb procedures on performance.

C.1 Lower bound procedure of Ng and Mak

For their single crane yard crane scheduling problem as presented in Ng and Mak (2005) they provide a lower bound procedure.

C.2 Performance

To measure the performance of the sublb procedures, we create 100 problems where a single yard crane has to perform 25 jobs in a range of 40 bays. We use the same construction op problems as in Chapter 9. The results are presented in Table C.1. Finding an optimal solution within 120 seconds was not possible for 6 problem instances.

In those cases we presented the best found solution by the algorithm solving the single crane problem.

Characteristics	Runtime (s)	Objective (s)	#out of time
Exact	13.0826	4.4360	6
sublb1	0.0001	3.2311	0
sublb2	0.0196	2.4055	0
sublb3	0.0134	1.6296	0
sublb4	0.0571	3.4441	0

Table C.1: Runtime and objective for sublb procedures

The sublb procedures found in 100% of the problem instances a lower bound within 50% of the solution found by the single crane algorithm. In 66% of the problem instances, they found a solution within 25% and in 19% of the problem instances wihtin 10%. From Table C.1 it becomes clear that sublb4 performs best with respect to the strength of the lower bound (a strong lower bound is close to the acual value). Based on computation times, sublb1 performs best. Sublb1 finds a decent lower bound in many cases which makes it very suitable for the Benders decomposition algorithm. Calling for sublb1 to find a lower bound does cost much compuation time and in 43% of the problem instances, sublb1 found a lower bound witin 25% of the exact solution. In 93% of the cases, sublb1 finds a lower bound within 50% of the solution found by the single crane algorithm. Assuming these single crane problems created by the Benders decomposition will lead to much delay and will therefore be discarded as a solution, a solution within 50% is already likely to be discarded.

In 22 out of 57 cases that sublb1 did not find a lower bound within 25%, sublb2 found a stronger lower bound than sublb1. In 9 of these 22 problem instances, sublb2 did find a lower bound within 25% of the exact solution. This justifies the use of the sublb2 in the decomposition algorithm. The computation times for sublb2 are relatively short compared to solving the single crane problem so if can be avoided to solve the single crane problem, it is almost always worth it. In 5 out of 57 cases, sublb3 found a lower bound that was stronger than sublb1. However, in all those cases, sublb2 was stronger than sublb3.

In 23 out of 100 cases, it took Gurobi longer than 10 seconds to solve the singel crane problem. In 13 out of 23 of those cases, sublb1 did find a lower bound within 25% of the exact solution. In 16 out of 23 cases, sublb4 did find a lower bound within 25% of the optimal solution. The runtime of sublb4 was always between 0.05 and 0.07 seconds. Sublb 4 always finds a stronger lower bound than sublb1. Instead of ignoring the travel time of the yard cranes, sublb4 includes a lower bound on those travel times. Therefore, the completion times of the jobs is always larger in sublb4 than in sublb1. This will lead to a stronger lower bound. In 12 out of the 57 cases that sublb1 did not find a lower bound within 25% of the exact solution, sublb4 did find a lower bound within 25% of the exact solution.

Comparing sublb2 with sublb4, sublb2 finds in 21 out of 100 problem instances a stronger lower bound than sublb4. While sublb4 finds lower bounds that are stronger but always close to the lower bounds of sublb1, sublb2 can find much stronger lower bounds or much weaker lower bounds. If we use two sublb procedures, it is more advantageous to use sublb procedures that produce very different results above sublb procedures that find similar results.