Erasmus University Rotterdam

Bachelor Thesis

Resource Optimisation at Maritime Oil Terminals

Author:
John Brouwer

Supervisor:
Dr. Adriana Gabor

June 25, 2013

Abstract

The problem we consider in this thesis is the discrete dynamic berth allocation problem with integrated pipeline assignment planning. In this problem processing speed of oil batches depend on the chosen set of pipeline segments to the goal tank as pipeline segments may have different maximum pumping speeds. Cleaning time can also be taken into account if this is needed when sequentially processing different oil products through the same pipeline segment.

We successfully come up with a linear mixed integer programming formulation to describe the considered problem. We show that for cases of realistic sizes CPLEX may not be able to solve the problem to optimality. Selecting the strong branching variable selection technique results in lower upper bounds if the problem cannot be solved to optimality. Strong branching mostly results in optimality faster too.

The heuristics that we implement are the benchmark first-come, first-served (FCFS) heuristic and squeaky wheel optimisation to further improve on the initial FCFS solution. A comparison of the heuristics and the exact method with seven test cases shows that CPLEX with strong branching results in better upper bounds but sometimes also needs more than an hour to solve to optimality. The heuristics on the other hand run in negligible time.

Contents

Chapter 1

Introduction

Maritime oil terminals have to cope with difficult planning decisions. Not only do they need to assign all vessels to a compatible berth and a set of pipeline segments such that the vessel can unload all the batches into the specified tanks, but ideally a schedule also minimises the total amount of time vessels lay in the harbour, either waiting or being handled. The reason for this is that the owner of a vessel can charge very high costs for the time it has to spend in the harbour and cannot do other shipping tasks.

Berth and pipeline assignment at maritime oil terminals is until now mostly done by hand. This means that a big decrease in costs may be achieved by planning optimally or close to optimal. Therefore our research consists of finding an exact formulation to optimise resources and developing heuristics for when running times of the exact method are too high.

1.1 Problem definition

For a certain time period in the future a set of arriving vessels is given. Of each vessel we know the arrival time, with which berthing places it is compatible, the batches with type of oil, amount, and what tank to pump it into, and the maximum pumping speed the vessel allows. The batches have to be processed in the given sequence and processing of two batches of the same vessel may not be done simultaneously. To the berthing places a system of pipelines is connected of which we know for each segment what the maximum pumping speed is.

Every berthing place can only handle one vessel at a time. The same holds for a pipeline segment. The speed with which a batch is pumped to the right tank is the minimum of the maximum pumping speed of every used pipeline segment and the maximum pumping speed of the vessel. Cleaning time of pipeline segments has to be taken into account when certain products are pumped through a pipeline segment which had another type of product pumped through it before. The combinations of products that would require cleaning in between are given.

The decision that has to be made is which vessel to assign to which compatible berthing place, what series of pipeline segments to use for processing the batches, and when the batches should start being processed. The goal is to minimise the total delay time (waiting plus handling time) of all vessels.

1.2 Literature review

When solving problems like stated above we speak of the berth allocation problem (BAP). This problem has intensively been studied for container terminals, as nicely summarized in Bierwirth and Meisel (2010), but not for bulk and oil terminals. At bulk ports the handling time of a vessel is dependent on the assigned berth as opposed to container terminals, where in most formulations a sufficient number of quay cranes is ensured but decreasing handing times with even more quay cranes is not modelled. Therefore a slightly different formulation is used for bulk ports.

There are different types of BAPs. The main division lies in the arrival process of vessels:

- SBAP: In the static variant it is assumed that all vessels have arrived at the beginning of the planning period and therefore arrival times do not have to be taken into account.
- DBAP: In the dynamic variant the vessels arrive throughout the planning period and a vessel cannot be assigned to a berth before it has arrived.

The division of the quay also makes a difference:

- Discrete BAP: Here the quay consists of a set of completely separate berths. Any berth can only be used by one vessel at a time.
- Continuous BAP: We speak of a continuous BAP when a vessel can berth at any position along the quay that is not already in use.
- Hybrid BAP: In the hybrid BAP the quay consists of sections that may have multiple vessels at a time assigned to them. Also, a vessel can occupy multiple sections at a time. The advantage of the hybrid BAP is that restrictions on for example the draft of the vessel and height of the quay can be imposed.

Because we are interested in the discrete DBAP formulations we take a closer look at literature that discusses those. Buhrkal et al. (2011) compares three models to solve the discrete DBAP. The first one comes from Imai et al. (2001), the second one from Cordeau et al. (2005), and the last one from Christensen and Holst (2008).

The method described in Christensen and Holst (2008) is a generalized set-partitioning problem with time intervals. The optimisation model of this method is very easy because feasible assignments are created beforehand. Such an assignment shows what vessel is considered and to what 'berth-time interval'

combinations it is assigned in the specific feasible assignment. The advantage of this method is that the matrix with all feasible assignments is split up in such a way that one set of constraints acts as a set of generalized upper bound constraints. Such constraints result in a feasible region of which all extreme points are integers and therefore decreases running times dramatically.

Imai et al. (2001) and Cordeau et al. (2005) both control the assignment of vessels to berths as well as the order in which vessels are assigned to a certain berth. The main difference between the formulations is that Imai et al. (2001) ensures that handling of vessels on the same berth does not overlap through constraints defined in the model. Cordeau et al. (2005) on the other hand views the problem as a multi-depot vehicle routing problem with time windows (MDVRPTW), where berths are vehicles and vessels are customers. Therefore, ensuring that handling of vessels on the same berth does not overlap has to be done through the decision of whether or not to create certain edges in the routing graph. Buhrkal et al. (2011) slightly improves both of the formulations discussed in this paragraph in order to simplify and decrease running times without losing functionality.

Continuous and hybrid DBAP formulations are discussed by Umang et al. (2012) for example. It can easily be seen that the discrete DBAP is just a special case of the hybrid DBAP and therefore using the hybrid DBAP formulation for a discrete problem is perfectly possible.

We decide to also look at formulations for container terminals where the quay crane assignment problem is integrated with the berth allocation problem. This may give us ideas on how to integrate our pipeline assignment problem into the berth allocation problem. After inspecting Bierwirth and Meisel (2010) we decide to look at Imai et al. (2008). This article proposes a formulation for simultaneous berth and quay crane assignment.

With respect to the development of heuristics a very straightforward first-come, first-served heuristic is proposed in Buhrkal et al. (2011). A slightly more complex heuristic is discussed in Umang et al. (2012). Squeaky wheel optimisation (SWO) is used to try and improve on the initial first-come, first-served solution. This is done by analysing the contribution of every vessel to the goal value and basing a new prioritisation on that, which means that the vessel with the highest sum of waiting and handling time in the initial solution gets assigned first in the second iteration and the vessel with the lowest sum of waiting and handling time gets assigned last. This procedure should iteratively decrease the goal value and Umang et al. (2012) claims that this method gives much better solutions (closer to optimal) than the first-come, first-served solution within a minute running time. For fundamental information on SWO we refer to Joslin and Clements (1999).

1.3 Methodology

Because handling times are dependent on our choices it seems natural to create a formulation like the ones for bulk ports. The situation we analyse is even slightly different from a normal bulk port though, because the handling times do not necessarily depend on the assigned berth but on the assigned pipelines. Therefore, we look at both formulations for bulk ports and container terminals and decide in which the pipeline assignment is easiest integrated.

Despite the short running times of and compact optimisation model behind the method proposed in Christensen and Holst (2008) we cannot use this method because in our case there is a restriction involving the subsequent pumping of different products through the same pipeline segment. This means that the feasibility of an assignment of one vessel depends on the chosen assignment of another vessel. Such constraints are very hard to model with this formulation because this formulation can only ensure that all vessels are assigned once and in each time interval a berth does not have multiple assigned vessels.

Both the formulations from Imai et al. (2001) and Cordeau et al. (2005) can easily be rewritten to fit the BAP subproblem of our problem. One thing that for example would have to be added is the restriction that some vessels may not be able to berth everywhere, another that one vessel may contain multiple batches and that the handling time of the vessel is the ending time of processing of the last batch minus the starting time of handling of the vessel. Still, we do not prefer to use those formulations because some sets of constraints become non-linear when the handling times are decision variables. This is true for our problem as the handling times depend on the pipeline assignment decisions. Unfortunately, the same is true for the formulation of Imai et al. (2008), so we also prefer not to use that formulation.

The only DBAP formulation that we have looked at in the literature review that does not give sets of non-linear constraints when the handling times become decision variables is the hybrid DBAP formulation proposed by Umang et al. (2012). For that reason we transform the hybrid DBAP formulation into a discrete DBAP formulation and integrate the pipeline assignment problem in chapter 2. In that chapter we also run seven test cases to assess the performance of the exact method with different settings. All tests are run on a 64-bit Intel® Core^{TM} i3-330M with 4GB of RAM.

We implement both the first-come, first-served heuristic and the squeaky wheel optimisation in chapter 3. In our case we do not have to take into account different berth opening times and therefore the first-come, first-served heuristic would come down to prioritising vessels on their arrival times. Allocation can then be based on waiting and handling times of compatible berths. At the end of chapter 3 we compare the developed heuristics with each other and the results from the exact method. This will again be done by means of the seven test cases.

Chapter 2

Exact method

In this chapter we will model the discussed problem as a mixed integer program. The basic formulation will be discussed in section 2.1. In section 2.2 we expand the formulation to also cover maintenance tasks to berths. Section 2.3 covers a slightly different problem from the one presented above, where also pipeline intersections cannot be used for simultaneously processing different batches. In section 2.4 we go into the details of the implementation of this model and results and discussion of testing are shown in section 2.5.

2.1 Basic formulation

The basis of the way we model the pipeline system is a set of pipeline intersections. Pipeline intersections are connected to other intersections through pipeline segments. This means that berthing places and tanks are also intersections because they are connected to the rest of the pipeline network. One pipeline intersection could be connected to multiple other pipeline intersections. One can compare pipeline intersections with traffic junctions connecting roads to form a road network. For the exact formulation of this problem we use the following sets:

B	Set of berths.
K	Set of pipeline intersections (where $B \subset K$).
L_f	Set of pipeline intersections to which can be pumped from intersection f, for all $f \in K$.
T	Set of cargo types.
V	Set of vessels.
W_i	Set of batches on board vessel $i : \{1, 2, \ldots, N_i\}$, for all $i \in V$.

We also make use of the following parameters:

a_i	Arrival time of vessel i, for all $i \in V$.
b_{ij}	Binary parameter of value 1 if vessel i can berth at berth j and 0 otherwise, for all $(i, j) \in V \times B$.
$d_{t_1 t_2}$	Binary parameter of value 1 if cleaning time is needed when cargo type t_2 is pumped through a pipeline segment right after cargo type t_1 has been pumped through the same segment and 0 otherwise, for all $(t_1, t_2) \in T^2$.
M	A very large number.
g_{iw}	Pipeline intersection at which the tank is located to which batch w of vessel i should be pumped, for all $(i, w) \in V \times W_i$.
q_{iw}	Quantity of batch w of vessel i, for all $(i, w) \in V \times W_i$.
t_{iw}	Cargo type of batch w of vessel i, for all $(i, w) \in V \times W_i$.
v_i	Maximum pumping speed of vessel i (in units of time per units of volume), for all $i \in V$.
v_{fg}	Maximum pumping speed of pipeline segment between intersections f and g (in units of time per untis of volume), for all $(f, g) \in K \times L_f$.

Next to that we use the following decision variables:

s_i	Starting time of handling of vessel i, for all $i \in V$.
c_i	Total handling time of vessel i, for all $i \in V$.
x_{ij}	Binary variable of value 1 if vessel i is assigned to berth j and 0 otherwise, for all $(i, j) \in V \times B$.
$\phi_{ii'}$	Binary variable of value 1 if vessel i' is handled after vessel i has been handled and 0 otherwise, for all $(i, i') \in V^2$ where $i \neq i'$.
s_{iw}	Starting time of handling of batch w of vessel i, for all $(i, w) \in V \times W_i$.
c_{iw}	Handling time of batch w of vessel i, for all $(i, w) \in V \times W_i$.
y_{iwfg}	Binary variable of value 1 if the pipeline segment between intersections f and g is used for processing of batch w of vessel i in the direction of intersection g and 0 otherwise, for all $(i, w, f, g) \in V \times W_i \times K \times L_f$.
$\delta_{iwi'w'}$	Binary variable of value 1 if handling of batch w' of vessel i' starts after batch w of vessel i has been handled and 0 otherwise, for all $(i, i', w, w') \in V^2 \times W_i \times W_{i'}$ where $(i, w) \neq (i', w')$.
p_{iw}	Pumping speed of batch w of vessel i (in units of time per units of volume), for all $(i, w) \in V \times W_i$.

The formulation is as follows:

$$\min \quad \sum_{i\in V}(s_i - a_i + c_i), \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j\in B} b_{ij}x_{ij} = 1, \quad \forall\ i \in V, \tag{2.2}$$

$$s_{i'} + M(1-\phi_{ii'}) \geq s_i + c_i, \quad \forall\ (i,i') \in V^2, i \neq i', \tag{2.3}$$

$$2 - x_{ij} - x_{i'j} + \phi_{ii'} + \phi_{i'i} \geq 1, \quad \forall\ (i,i',j) \in V^2 \times B, i' < i, \tag{2.4}$$

$$s_{iw} \geq s_{iw-1} + c_{iw-1}, \quad \forall\ (i,w) \in V \times W_i, w > 1, \tag{2.5}$$

$$\sum_{f\in L_j} y_{iwjf} = x_{ij}, \quad \forall\ (i,w,j) \in V \times W_i \times B, \tag{2.6}$$

$$\sum_{g\in L_f} y_{iwfg} - \sum_{g:f\in L_g} y_{iwgf} = 0, \tag{2.7}$$
$$\forall\ (i,w,f) \in V \times W_i \times (K \setminus \{B, g_{iw}\}),$$

$$\sum_{g\in L_f} y_{iwfg} \leq 1, \quad \forall\ (i,w,f) \in V \times W_i \times (K \setminus B), \tag{2.8}$$

$$\sum_{f:g_{iw}\in L_f} y_{iwfg_{iw}} = 1, \quad \forall\ (i,w) \in V \times W_i, \tag{2.9}$$

$$s_{i'w'} + M(1-\delta_{iwi'w'}) \geq s_{iw} + c_{iw} + 2d_{t_{iw}t_{i'w'}}, \tag{2.10}$$
$$\forall\ (i,i',w,w') \in V^2 \times W_i \times W_{i'}, (i,w) \neq (i',w'),$$

$$2 - y_{iwfg} - y_{i'w'fg} + \delta_{iwi'w'} + \delta_{i'w'iw} \geq 1, \tag{2.11}$$
$$\forall\ (i,i',w,w',f,g) \in V^2 \times W_i \times W_{i'} \times K \times L_f, (i',w') < (i,w),$$
$$\text{when } f \notin L_g$$

$$2 - y_{iwfg} - y_{iwgf} - y_{i'w'fg} - y_{i'w'gf} + \delta_{iwi'w'} + \delta_{i'w'iw} \geq 1, \tag{2.12}$$
$$\forall\ (i,i',w,w',f,g) \in V^2 \times W_i \times W_{i'} \times K \times L_f, (i',w') < (i,w),$$
$$\text{when } f \in L_g$$

$$p_{iw} \geq v_i, \quad \forall\ (i,w) \in V \times W_i, \tag{2.13}$$

$$p_{iw} \geq v_{fg}y_{iwfg}, \quad \forall\ (i,w,f,g) \in V \times W_i \times K \times L_f, \tag{2.14}$$

$$c_{iw} \geq p_{iw}q_{iw}, \quad \forall\ (i,w) \in V \times W_i, \tag{2.15}$$

$$s_i \geq a_i, \quad \forall\ i \in V, \tag{2.16}$$

$$s_{i1} \geq s_i, \quad \forall\ i \in V, \tag{2.17}$$

$$s_i + c_i \geq s_{iN_i} + c_{iN_i}, \quad \forall\ i \in V, \tag{2.18}$$

$$x_{ij} \in \{0,1\}, \quad \forall\ (i,j) \in V \times B, \tag{2.19}$$

$$\phi_{ii'} \in \{0,1\}, \quad \forall\ (i,i') \in V^2, i \neq i', \tag{2.20}$$

$$y_{iwfg} \in \{0,1\}, \quad \forall\ (i,w,f,g) \in V \times W_i \times K \times L_f, \tag{2.21}$$

$$\delta_{iwi'w'} \in \{0,1\}, \tag{2.22}$$
$$\forall\ (i,i',w,w') \in V^2 \times W_i \times W_{i'}, (i,w) \neq (i',w'),$$

The objective function (2.1) that we minimise represents the waiting and handling time of all vessels together. Constraints (2.2)–(2.4) and (2.16) take care of the berth allocation problem. Constraints (2.5) ensure that processing of batches of the same vessel cannot be done simultaneously and should be done in the given order. Constraints (2.6)–(2.9) are flow conservation constraints of pipeline allocation. Constraint sets (2.10)–(2.12) ensure that processing of batches over the same pipeline segment is not done simultaneously. Constraints (2.13)–(2.15) make sure the pumping speed and handling time of every batch are defined such that unloading the batch is possible with the chosen berth and pipeline segments. Constraints (2.17)–(2.18) connect the vessel level of this problem to the batch level.

In more detail, constraints (2.2) ensure that every vessel is assigned to exactly one berth that is compatible with that vessel. Constraints (2.3) control the definition of $\phi_{ii'}$, which is further used in set (2.4). Constraint set (2.4) ensures that two different vessels are processed at different berths or after the other has finished or both, because when at least one of a pair of vessels is not assigned to a berth the constraint is immediately satisfied. When both of a pair of vessels are assigned to the same berth one vessel has to start handling after the other has finished to satisfy the constraint. Constraint set (2.16) makes sure that the starting time of the handling of a vessel is not before it has arrived.

Constraint set (2.5) makes sure that batches of a given vessel are processed in the specified sequence. This also ensures for batches of the same vessel that they are not processed simultaneously.

Constraints (2.6) ensure that the total flow coming from a berth to unload a batch of a given vessel is equal to 1 if that vessel is berthed there and 0 otherwise. Note that we have defined the sets L_f (for all $f \in K$) in such a way that, assuming pumping towards a berth is not possible, we do not have to look at the y-variables of pipeline segments going to berths as they are not defined. Constraints (2.7) make sure that no flow is lost and constraints (2.8) ensure that there are no cycles in the pipeline allocation as this would mean pumping through the same pipeline segment twice for one batch. Constraint set (2.9) makes sure that the total flow going to the goal tank of a batch is equal to 1. Here we have also made use of the fact that y-variables of pipeline intersections to which tanks are connected are only defined towards the tank and not back.

Constraint set (2.10) controls the definition of $\delta_{iwi'w'}$, which is further used in constraint sets (2.11) and (2.12). Sets (2.11) and (2.12) ensure for every pipeline segment that it is not used for two batches at the same time. Constraints (2.11) do so for pipeline segments that are not reversible, so do not allow processing in two directions. When a segment is used for both of a pair of batches one batch must start handling after the other has finished (including 2 units of cleaning time in between if necessary). Constraints (2.12) prevent simultaneous processing over a reversible pipeline segment by taking into account both flow directions. One batch will never have flow in two directions over the same pipeline segment and therefore if one pipeline segment is used by both of a pair of batches, either in the same direction or opposing directions, a solution is still feasible if one batch starts handling after the other has finished. Note

that cleaning time is always counted even though a batch may be processed in between the two batches at which is being looked. This may theoretically result in infeasible solutions that should be feasible. Therefore we assume that if cleaning time is needed between two products it is not possible that the need to clean can be undone by processing a different product in between.

Constraint set (2.13) ensures that the pumping speed of a certain batch, in units of time per units of volume, is at least the speed the vessel allows and constraints (2.14) do the same for all used pipeline segments. Constraints (2.15) make sure that the handling time of a batch is high enough to unload everything at maximum speed. Note that multiplying by the speed instead of dividing by, which would render the constraints non-linear, is possible because we always defined the speed in units of time per units of volume. This is the inverse of what would normally be called the speed.

Constraints (2.17) make sure that the starting time of handling of the first batch of a given vessel is not earlier than the starting time of handling of the given vessel. Constraint set (2.18) ensures that the total handling time of a vessel is at least the time between the starting time of handling of the vessel and the ending time of handling of the last batch.

2.2 Maintenance tasks

It is possible that one also wants to take into account maintenance of berths. In such case the berthing place cannot be used by any vessels for a given period of time. To add this to the model we need the additional set R which holds all the maintenance tasks. We declare the following parameters:

l_{rj}	Binary parameter of value 1 if maintenance task is performed on berth j and 0 otherwise, for all $(r, j) \in R \times B$.
n_r	Starting time of maintenance task r, for all $r \in R$.
o_r	Ending time of maintenance task r, for all $r \in R$.

We also need the following additional decision variables:

β_{ir}	Binary variable of value 1 if handling of vessel i starts after maintenance task r has finished and 0 otherwise, for all $(i, r) \in V \times R$.
β_{ri}	Binary variable of value 1 if maintenance task r starts after vessel i has been handled and 0 otherwise, for all $(r, i) \in R \times V$.

To successfully integrate maintenance tasks we now add the following sets of constraints to the model presented in section 2.1:

$$s_i + M(1 - \beta_{ir}) \geq o_r, \quad \forall\ (i, r) \in V \times R, \tag{2.23}$$

$$n_r + M(1 - \beta_{ri}) \geq s_i + c_i, \quad \forall\ (r, i) \in R \times V, \tag{2.24}$$

$$2 - l_{rj} - x_{ij} + \beta_{ir} + \beta_{ri} \geq 1, \quad \forall\ (i, j, r) \in V \times B \times R, \tag{2.25}$$

As shown above we have now ensured that if a vessel is assigned to a berth on which a maintenance task is performed the vessel must be handled before or after the maintenance task is scheduled.

2.3 Simultaneous intersection usage

In this thesis we also want to provide a formulation for a problem closely related to the problem that we already talked about. Until now we allowed processing of multiple batches over the same intersection simultaneously as long as they did not use the same pipeline segments. In some applications though there may not lie restrictions only on simultaneous usage of pipeline segments but also of pipeline intersections. Our assumption is that this means that cleaning time also depends on the used pipeline intersections. To transform the problem for such cases a couple of changes have to be made.

Firstly, an additional set of binary decision variables z_{iwf} will have to be introduced. Such binary variable is 1 if batch w of vessel i is processed through pipeline intersection f and 0 otherwise, for all $(i, w, f) \in V \times W_i \times K$.

Second, the restriction $z_{iwf} \geq y_{iwgf}$ will have to be introduced for all $(i, w, f, g) \in V \times W_i \times K \times (g : f \in L_g)$. Furthermore, restriction sets (2.11) and (2.12) should be replaced by $2 - z_{iwf} - z_{i'w'f} + \delta_{iwi'w'} + \delta_{i'w'iw} \geq 1$ for all $(i, i', w, w', f) \in V^2 \times W_i \times W_{i'} \times K, (i', w') < (i, w)$. All z-variables also have to be restricted to binary values.

As can be seen from the set above now one intersection cannot be used for processing of two batches at the same time while taking into account cleaning time.

It is also possible that one only wants to impose restrictions on simultaneous intersection usage for a subset of intersections. One practical example would be that some tanks are connected to multiple segments and simultaneously pumping batches of different vessels into the same tank is not allowed. In such case the restrictions presented in this section would give the right results when only imposed for all f in the set of tank intersections. When only restricting simultaneous intersection usage for a subset of intersections restriction sets (2.11) and (2.12) should not be replaced unlike stated before.

Because in our problem definition we have no restrictions on simultaneous pipeline intersection usage we will not implement this. We will however now implement the basic formulation with the maintenance tasks expansion.

2.4 Implementation

We implement the model from sections 2.1 and 2.2 in Java using the Callable Library of CPLEX. While most things are straightforward a couple of things are important to note.

First of all we have a couple of sets of constraints which use the so-called big-M. While in the formulation this is the best way to keep the model linear it

may give computational problems. To start with, the chosen value of M has a certain lower bound under which the constraint will not have the desired effect. On the other hand, very large values of M create two disadvantages as well. Firstly, upper bounds on the problem will be very loose with large values of M. This significantly increases computational times. Secondly, and most important, because of the way numbers are represented on a computer infeasible solutions may be seen as feasible solutions. This is often called trickle flow.

In CPLEX there are workarounds to the big-M formulations while keeping the constraints linear. One of these is to use indicator constraints like if-then statements. This way the control over a binary variable can be determined by whether a different expression is true or not, and the other way around. Apart from indicator constraints to represent big-M formulations, logical constraints can also be used. If we would have formulated the constraint $b + M(1 - x) \geq a$, where a and b can be any type of decision variable and x a binary decision variable, the logical constraint would be $x <= (b >= a)$. This way x must be zero if b is lower than a and can take any binary value otherwise. Although such constraints are not linear by themselves CPLEX automatically turns them into linear ones if no quadratic terms appear in the arguments of the expressions. CPLEX does this by creating additional variables and constraints.

Apart from computational tricks some practical tricks may also be needed to use this formulation. One assumption that is made, for example, is that the time between two vessels being assigned to the same berth can be zero. This is practically impossible as unmooring of one ship and mooring of another will take some time. Things like these, either on the vessel level or batch level, are very easily implemented. On the vessel level one can add a fixed number of units of time to the right-hand side of constraint set (2.3) to make sure that amount of time will always be between any two vessels being assigned to the same berth. On the batch level the same holds for constraints (2.5). It is even possible to make sure there is time between the starting time of handling of the vessel and the starting time of handling of its first batch. This can be done applying by the same trick as explained above to set (2.17). To add extra handling time after handling of the last batch the trick can be applied to set (2.18).

2.5 Results

We create seven test cases with which we can assess the performance of the exact method. The description of these cases are given in table 2.2. We run CPLEX with standard settings but also try strong branching as alternative variable selection method. Strong branching performs more in-depth exploration and therefore builds up the tree size slower.

The results of the performance tests are shown in table 2.3. Note that we run the optimisation until a maximum runtime of one hour. If the solution has not been proved to be optimal but the reported runtime is under one hour it means that optimisation terminates due to an out-of-memory status of CPLEX.

Case	Description
1	The pipeline system is a grid of two segments long and two segments wide. There are four berths, which are all compatible with all vessels, and three tanks, evenly distributed as goal tanks over the batches. All vessels and segments have pumping speed 1 and there is one maintenance task on the third berth from time 4 until 6. There are 8 vessels with arrival time 0 for vessel 1, 1 for vessel 2, 2 for vessel 3, and so forth.
2	The same as case 1 except for the arrival times. The arrival times are drawn from a uniform distribution with bounds [0,10]. The ordered vessel arrival times are 0.3, 2.6, 4.1, 6.3, 7.7, 7.9, 7.9, and 9.1.
3	The same as case 1 except for the arrival times. The arrival times are drawn from a uniform distribution with bounds [0,10]. The ordered vessel arrival times are 0.1, 3.9, 5.3, 5.6, 5.7, 6.2, 8.4, and 9.5.
4	The same as case 2 except batches of vessel 6 (with arrival time 7.9) are of size 4 and batches of vessel 7 (with arrival time 7.9) of size 7.
5	The same as case 2 except vessel 6 (with arrival time 7.9) pumps twice as slow as the other vessels and vessel 5 (with arrival time 7.7) four times as slow as the other vessels.
6	The same as case 3 except batches of vessel 4 (with arrival time 5.6) are of size 10 and batches of vessel 5 (with arrival time 5.7) of size 5.
7	The same as case 3 except vessel 4 (with arrival time 5.6) pumps five times as slow as the other vessels and vessel 5 (with arrival time 5.7) ten times as slow as the other vessels.

Table 2.2: Description of seven test cases.

Case	Standard variable selection			Strong branching		
no.	UB	LB	Runtime (sec)	UB	LB	Runtime (sec)
1	56.0	39.0	1958	54.0	54.0	171
2	52.3	41.5	2643	50.1	50.1	1575
3	59.7	35.6	1537	58.1	53.3	3600
4	69.5	59.3	3600	69.5	69.5	203
5	71.5	64.9	3600	71.5	71.5	92
6	85.1	65.7	3229	85.1	85.1	477
7	128.0	123.9	3600	128.0	128.0	54

Table 2.3: Results of exact method on seven test cases.

With standard settings four out of seven test cases (57%) result in an out-of-memory error. The three other cases run for at least an hour so we do not know if they could be solved eventually. With strong branching no case results in an out-of-memory status (0%) and we only terminated solving of case 3 because it took longer than one hour. From this we can conclude that setting the variable selection method to strong branching decreases the chance of termination due to an out-of-memory status.

Of the three cases that did not result in an out-of-memory error within one hour with standard variable selection (case 4, 5, and 7) it was case 4 that took longest to solve with strong branching. Strong branching only needed 203 seconds though, which is under 6% of the runtime with standard settings. This means that strong branching clearly needs less time than the standard variable selection method to solve to optimality.

Although the amount of cases is not high enough to perform statistical tests, we can see two main influences on running times by looking at the running times of the different test cases.

- Inter-arrival times: The first influence is how close together the vessels arrive. This is logical because if there is a long time between two arriving vessels the choice of what vessel to serve first is much easier. This is clearly illustrated by the difference in running time between case 1 and cases 2 and 3.

- Heterogeneity of vessel properties: The second influence is what we call the heterogeneity of vessels. When the inter-arrival time between two vessels is very small and the vessels have the same properties it takes a long time to decide on what vessel to handle first. When one vessel has very different properties the choice is much easier. These properties can either be the size of the batches, as shown by the differences in running times between cases 2 and 4 and between cases 3 and 6, or the maximum pumping speed of the vessels, supported by the differences in running times between cases 2 and 5 and between cases 3 and 7.

Despite the fact, due to limited time, that we have not been able to support this with test cases, we can also logically derive that the size of the pipeline system influences running times. A very small pipeline system mainly results in short running times because the amount of choices is small. As the pipeline system increases running times increase. We can illustrate this with a thought experiment by comparing one of the cases from table 2.2 with one just like that except for the pipeline system being just one pipeline segment connecting the segments from the various berths to the segments to the various tanks. In the latter case the only thing we can try to optimise is the handling order of vessels as there is no possibility to choose a different set of pipeline segments. Very large pipeline systems allow for more simultaneous processing and therefore cases with such systems are solved quicker again. In this case they will be solved quicker because a low upper bound is more easily found.

Finally, and also trivial, if additional vessels are included in a planning assignment running times will never decrease as no decisions of the former planning assignment will be easier when another vessel is included.

Because some discussed cases are not solved to optimality using the exact formulation we will focus on heuristic methods in the next chapter.

Chapter 3

Heuristic methods

In this chapter we propose two heuristic methods: first-come, first-served and squeaky wheel optimisation. The first method will serve as a benchmark for the other one and may be valuable as a benchmark in later research too. In section 3.1 we describe the heuristics in further detail and in section 3.2 we show insight in the implementation. We compare the results of the two heuristic methods with the ones of the exact formulation in section 3.3.

3.1 Description of methods

3.1.1 First-come, first-served heuristic

The most basic heuristic that we implement is the first-come, first-served heuristic. In this heuristic the priorities of vessels are based on arrival times. This is basically what a human would also first try if getting the task of making a planning. If two vessels arrive at the same time prioritisation could be based on for example the amount of batches a vessel has or the total quantity of oil on board of a vessel. Which prioritisation is best is impossible to say because for each rule an example can be found where it is not beneficial. We assign a vessel to the berth that will be able to minimise the total delay of the vessel (waiting plus handling time).

Batches of a vessel are assigned in the given sequence. Pipeline segments are assigned greedy from the list of available pipeline segments. This means that the batch will be assigned to the path, which is a sequence of pipeline segments that connect the berth to the tank, that could minimise the ending time of processing of the batch (starting plus handling time). In case taking into account cleaning time for different types of cargo is too difficult one could simplify this part of the algorithm by taking into account cleaning time between every batch. That this leads to a higher upper bound on the objective value is trivial. For this reason we will only count cleaning time when needed.

3.1.2 Squeaky wheel optimisation

In the squeaky wheel optimisation (SWO) we work according to a construct-analyse-prioritise cycle. The initial planning that we construct is made by the first-come, first-served heuristic. After doing this we analyse for every vessel how large its contribution is to the total waiting and handling time of all vessels. The prioritisation of vessels in the next iteration is then based on that contribution to the objective value. Vessels that contributed a lot to the total waiting and handling time will have a higher priority in the next iteration than vessels that contributed less. This process of analysing the solution of the iteration before and then reconstructing a solution with prioritisation based on that analysis is repeated a specified number of times. To avoid getting trapped in a loop of already evaluated priority lists we randomly switch around places of a pair of neighbouring vessels in the priority list if that list has already been evaluated. After a specified number of iterations we choose the planning that had the best objective value. For this reason SWO will never perform worse than the FCFS heuristic.

A better method than the method that we have proposed to avoid looping over already evaluated priority lists might be to consider all priority lists that can be found by taking a random vessel and placing it somewhere else, and choosing the priority list that yields the best objective value. In that case if we have $|V|$ vessels the amount of candidate priority lists that we have to consider is $|V|(|V|-1)$ and we already know the next planning as it is one of the cases that we considered. If one of the candidate priority lists is already evaluated we may choose to leave it out and choose the best from all other candidates. It may occur though that all candidate priority lists have already been evaluated and to avoid getting trapped in a loop we would still have to come up with an alternative way out. Also in defence of random switching, disturbing the priority list by randomly switching a pair of neighbouring vessels is not a big problem as the mechanism behind SWO will undo any harm that has been done by the disturbance.

3.2 Implementation

Before running any of the heuristics an initialization procedure must be run to create all possible paths for every berth-tank combination. We choose this method because that procedure only has to be run once when running the heuristic. Creating these paths is easily done when using a recursion over the intersections and the segments connected to them.

For the FCFS heuristic we use the algorithm as shown in algorithm 1 (on page 17) and SWO is implemented as the pseudocode describes in algorithm 2 (on page 18). The prioritisation rule that we choose for FCFS in case vessels arrive at the same time is the hash code, an integer representation defined by Java, of the name of the vessel. We do this for now because we are not interested in observing how other prioritisation rules affect the objective value, and because

the hash code is unique as we have no vessels with equal names.

Unavailability of berths due to maintenance tasks, like we also considered in the exact formulation, is very easily implemented in these heuristics. Maintenance tasks can be considered by seeing them as a vessel with given starting and handling times and assigning them to the right berth before the planning is being made. In SWO it is important that the maintenance task is not put in the priority list but considered in every iteration.

```
1  for all vessels do
2  |  determine the priority it should have by counting how many other
   |  vessels should have a higher priority
3  |  save the priority of this vessel in the priority list
4  end
5  for all priorities on the priority list (increasing) do
6  |  determine which vessel has this priority
7  |  for all berths compatible with current vessel do
8  |  |  determine when this berth comes available from vessel arrival
   |  |  onwards
9  |  |  for all batches of current vessel do
10 |  |  |  if current batch is first batch then
11 |  |  |  |  earliest batch starting time ← time when berth comes
   |  |  |  |  available from vessel arrival onwards
12 |  |  |  else
13 |  |  |  |  earliest batch starting time ← time when former batch
   |  |  |  |  finishes processing
14 |  |  |  end
15 |  |  |  for all paths from this berth to goal tank of given batch do
16 |  |  |  |  determine when this path comes available from earliest
   |  |  |  |  batch starting time onwards
17 |  |  |  end
18 |  |  end
19 |  |  save earliest time when this berth can end service of current vessel
20 |  end
21 |  choose berth and paths that are capable of ending service for current
   |  vessel earliest
22 end
```

Algorithm 1: First come, first served heuristic

```
1  get the initial priority list by the prioritisation rules of the FCFS
   algorithm
2  add this priority list to the set of evaluated priority lists
3  current priority list ← initial priority list
4  current best objective value ← infinity
5  current best performing berth and pipeline assignment ← empty
6  while maximum number of iterations not exceeded do
7  |  get the current berth and pipeline assignment based on the current
   |  priority list by means of algorithm 3 (page 19)
8  |  if the objective value of the current berth and pipeline assignment is
   |  lower than the current best objective value then
9  |  |  current best objective value ← objective value of the current
   |  |  berth and pipeline assignment
10 |  |  current best performing berth and pipeline assignment ← current
   |  |  berth and pipeline assignment
11 |  end
12 |  for all vessels do
13 |  |  determine the total waiting and handling time of this vessel
14 |  end
15 |  sort the vessels in decreasing order of total waiting and handling time
   |  and base the new priority list on this
16 |  current priority list ← new priority list
17 |  while the current priority list is in the set of evaluated priority lists
   |  do
18 |  |  switch a random pair of consecutive vessels in the priority list
   |  |  around
19 |  |  current priority list ← reordered priority list
20 |  end
21 |  add the current priority list to the set of evaluated priority lists
22 end
```

Algorithm 2: Squeaky wheel optimisation (SWO)

```
1  for all priorities on the priority list (increasing) do
2      determine which vessel has this priority
3      for all berths compatible with current vessel do
4          starting time of current vessel ← vessel arrival time
5          indicator of found gap ← false
6          counter of explored gaps in berth's planning ← 0
7          while big enough gap for current vessel not found and not all gaps
           in berth's planning explored do
8              next vessel ← first vessel that starts handling after starting
               time of current vessel
9              determine for current vessel the quickest processing time after
               starting time
10             if processing of current vessel ends before next vessel is
               planned to start then
11                 indicator of found gap ← true
12             else
13                 starting time of current vessel ← time when next vessel
                   finishes handling
14             end
15         end
16         save earliest time when this berth can end service for current
           vessel
17     end
18     choose berth and paths that are capable of ending service for current
       vessel earliest
19 end
```

Algorithm 3: Assignment planning procedure during SWO

3.3 Comparison of heuristics and exact method

The test cases that we will use in this section are the same as in section 2.4. For a description of the cases we refer back to table 2.2 on page 12. We run both the FCFS heuristic and SWO for all 7 cases. We run SWO 2 times with 1500 iterations. FCFS solutions are always found within a second and two SWO runs take about 10 seconds.

The reason for running SWO multiple times is that SWO does not always give the same result. This is caused by the slight randomness due to randomly switching a pair of consecutive vessels if a priority list has already been evaluated. Because running times of SWO are negligible one could overcome the problem sketched above by setting the amount of iterations much higher. One should always remember though that the amount of iterations must never be larger than the amount of possible orderings of vessels (in case of $|V|$ vessels this is $|V|!$). Furthermore, setting the amount of iterations close to the amount of possible orderings of vessels may heavily increase running times because the

last few unevaluated priority lists may take long to find by randomly switching consecutive pairs of vessels.

Case	FCFS	SWO	Exact
1	15%	4%	0%
2	25%	16%	0%
3	20%	19%	9%
4	32%	11%	0%
5	49%	13%	0%
6	48%	20%	0%
7	84%	11%	0%

Table 3.1: Optimality gaps of heuristic methods and exact method (strong branching) based on best lower bound found by the exact method in section 2.5.

The results of the FCFS heuristic and SWO are given in table 3.1, where they are compared with the best upper bounds found by the exact method with strong branching. We compare these three optimisation methods with the best lower bound found by the exact method with strong branching by means of the optimality gap. This means that if we report 0% it is proved that the optimal solution has been found.

We see that in all test cases SWO was able to improve on the FCFS solution. The amount by which SWO is able to decrease the goal value with respect to FCFS differs a lot. From the reported results it seems that SWO is able improve on the FCFS solution more if vessel properties differ more. This seems logical as different prioritisations make more difference then. Because both heuristic methods require only a couple of seconds runtime we would always recommend using SWO instead of FCFS.

From the results in table 3.1 we can also conclude that SWO is never able to get really close to the best upper bound found by the exact method. Therefore there is a trade-off between speed and efficiency, where SWO solves within 10 seconds but the exact method has always given better results within an hour.

Chapter 4

Conclusion

In this chapter we present the main findings of our research. We also look at what future research in this field might focus at.

4.1 Main findings

In chapter 2 we came up with an exact formulation to describe the discrete berth allocation problem with integrated pipeline assignment planning. Furthermore, we have shown possible additions to the formulation like the ability to integrate maintenance tasks or prevent intersection usage by batches simultaneously. The formulation is linear but does contain a couple of constraint sets with big-M formulations. This is computationally not a big problem when using logical or indicator constraints in CPLEX. We saw that selecting strong branching as variable selection method is better in all cases we looked at. Strong branching resulted in optimality faster and gave lower upper bounds if it was not able to solve to optimality. Furthermore, we also discovered that how close together vessels arrive and how different their properties are can have a large influence on running times. When vessels arrive closer together running times increase. Especially if their properties, like batch sizes and maximum pumping speed, are very much alike this can cause problems in solving to optimality.

The heuristics that we implemented in chapter 3 are the first-come, first-served (FCFS) heuristic, which serves as a benchmark, and squeaky wheel optimisation (SWO). When one just looks at the best upper bound found we have to conclude that using the exact method with strong branching has always given the best results in our test cases. The added value of SWO with respect to the FCFS heuristic was always clearly present. When one also looks at the running times we see that the heuristics that we developed, SWO especially, may be preferable as running times are negligible.

4.2 Future research

The heuristics that we developed in this thesis only assign vessels to the berth that is able to minimise its waiting and handling time and do this in different order to try and improve the solution. On the other hand, there may also be ways to split the problem into a berth allocation problem and a pipeline assignment problem to come to good solutions quickly. Future research could focus on the development of such heuristics.

Furthermore, in this thesis we have assumed deterministic arrival times. Some articles in the field of berth and quay crane allocation have already come up with formulations to cover stochastic arrival times. It is important to see how ideas from those articles can be used in maritime oil terminals to create more robust berth and pipeline allocations. While the stochastic arrival times are probably easily integrated in our exact formulation using stochastic programming the running times would probably increase even further. Therefore this research would mainly focus on heuristics covering stochastic arrival times.

Finally, in the field of the economic lot-sizing problem research is being done on heuristics that provide better performance and more stable solutions for a rolling horizon. While heuristics developed for the economic lot-sizing problem may be unable to be converted to the berth and pipeline assignment problem it would be good if research is done on berth and pipeline assignment for a rolling horizon.

Bibliography

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

K. Buhrkal, S. Zuglian, S. Ropke, J. Larsen, and R. Lusby. Models for the discrete berth allocation problem: A computational comparison. *Transportation Research Part E*, 47(4):461–473, 2011.

C.G. Christensen and C.T. Holst. Berth Allocation in Container Terminals. Master's thesis, Department of Informatics and Mathematical Modelling, Technical University of Denmark, 2008. in Danish.

J.-F. Cordeau, G. Laporte, P. Legato, and L. Moccia. Models and Tabu Search Heuristics for the Berth-Allocation Problem. *Tranportation Science*, 39(4): 526–538, 2005.

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

A. Imai, H.C. Chen, E. Nishimura, and S. Papadimitriou. The simulataneous berth and quay crane allocation problem. *Transportation Research Part E*, 44:900–920, 2008.

D.E. Joslin and D.P. Clements. "Squeaky Wheel" Optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

N. Umang, M. Bierlaire, and I. Vacca. Exact and heuristic methods to solve the berth allocation problem in bulk ports. Technical report, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne, 2012.