The stockyard planning problem: heuristics for an efficient management of stockyard capacity

Author: D.W.F. Boissevain (424964)
Supervisor: dr. D. Galindo Pecin
Second assessor: dr. W. van den Heuvel

Ab Ovo & Erasmus School of Economics
Erasmus University Rotterdam, The Netherlands
Master's thesis Operations Research and Quantitative Logistics

December 11, 2020

Abstract

In this thesis, we consider the stockyard planning problem in which the goal is to find a schedule for incoming vessels and stockpiles that minimises total vessel demurrage cost. This problem is of relevance for planners in any dry bulk terminal environment, where the available space on both the quay and stockyard needs be utilised to its maximum capacity. We propose several heuristic approaches, which are evaluated by an exact model. Computational experiments using these methods shows that we are able to significantly reduce the costs over traditional planning methods with competitive solutions for smaller instances. Furthermore, an analysis is performed to find out which instance properties challenge the heuristic methods and to see under what policy it is beneficial to split stockpiles.

Keywords: Stockyard planning, Dry bulk terminals, Berth allocation, Stockyard allocation, Stacker-reclaimer scheduling, Construction Heuristic, Squeaky wheel optimisation, Genetic algorithm, Ant colony optimisation, Valid inequalities

Contents

1 Introduction

Over the last fifty years, the consumption of coal has risen from 16,000 in 1965 to 44,000 TWh in 2015 (World Bank, 2020). In this period, coal stayed the second-most consumed energy resource just below oil and above natural gas. The importance of coal has persisted through the years due to the fact that it is used as a primary resource in electricity production (US Energy Information Agency, 2020a). More specifically, 40% of global electricity is produced by coal plants, as it often obtains the lowest levelised cost of electricity in countries such as Germany (Fraunhofer ISE, 2018). Moreover, coal is vital in large-scale steel manufacturing, as alternatives to coal do not seem to possess the same capability as that of coal (Babich and Senk, 2013).

Although the importance of coal has remained the last fifty years, the US and countries in the EU have decreased their production significantly (Eurostat, 2020; US Energy Information Agency, 2011, 2020b). This results in coal being imported at an increasing scale, often from far away countries and transported over sea in between ports. These ports, however, have limited capacity and resources to handle incoming stockpiles. A port often has a limited quay space to berth vessels and a small number of machines to store stockpiles. Moreover, due to environmental and safety policies, stockyards next to the port are being limited in their capacity (The Economist, 2020). Combined with the fact that vessels have a high opportunity cost when leaving later than expected at an occupied port, the importance of efficiently using resources to handle an increasing volume of coal rises.

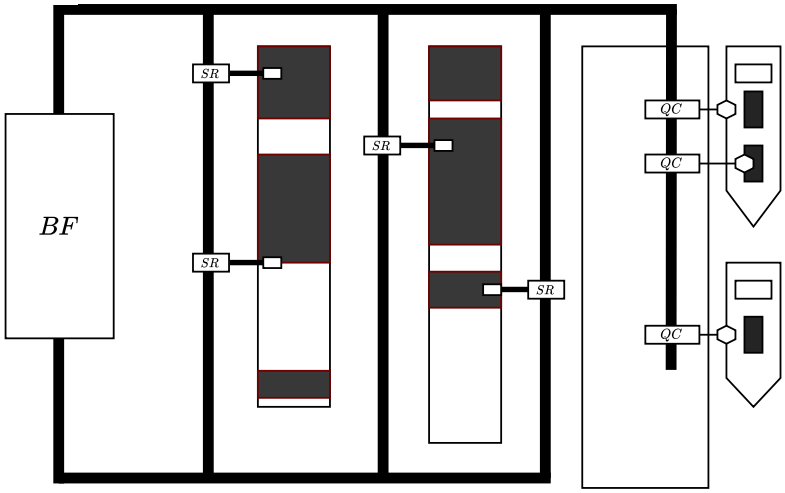


Figure 1: Overview of the harbor terminal with pads, quay cranes (QC), stacker-reclaimers (SR) and blast furnace (BF). The black solid line represents the conveyor belt and the solid rectangles within the pads represent the stockpiles.

This work serves to address the issue for a need in efficiency in a steel factory terminal port. In such a port, a production process of steel needs raw materials such as coal and iron ore. These raw materials are supplied by vessel, stored on a stockyard and consumed in the production process. An overview of this process is given in Figure 1. First, vessels arrive at a quay unloading their materials via quay cranes. During the unloading process, the raw material is moved from a harbor terminal to the stockyard by a conveyor belt, where it is stored as a stockpile. The stockyard consists of a number of coal and iron pads, which are long narrow inventory grounds on which stockpiles can be stored. Each coal and iron stockpile needs to be inserted into a pad before it can be used in the blast furnace. Each stockpile is stored such that the full width of a pad is used.

Stockpiles are built on and removed from the stockyard by a stacker-reclaimer. A stacker-reclaimer is a big crane-like machine which moves alongside the coal pad to 'stack' (create) and to 'reclaim' (remove) stockpiles to and from the stockyard. It can perform both these operations as it both has a conveyor belt to drop coal and a water wheel-like shovel wheel to reclaim a stockpile. When the blast furnace next to the stockyard requests coal or iron for its production, a part of a pile can be reclaimed making room for other stockpiles coming in.

We identify some decisions we have to make within the given context, which we call the stockyard planning problem (SPP). First, we have to decide where and when the vessels moor at the quay. Next, we need to choose a pad, position, and duration a stockpile is assigned to the stockyard. Lastly, we have to assign available stacker-reclaimers to a stockpile. Given these decisions, we can identify three subproblems within the SPP:

- Berth Allocation Problem (BAP) to determine a berth and berthing time for each vessel,
- Yard Allocation Problem (YAP) to determine the stockpile location and duration for each stockpile,
- Stacker-reclaimer Scheduling Problem (SRSP) to determine a schedule for each SR.

To solve the SPP, one could attempt to solve these three problems separately. First solving the BAP, then the YAP and then the SRSP. These three problems, however, are strongly interrelated. Stockpile placement strategies in the stockyard, for example, have a direct effect on the berthing options of the incoming vessels and vice versa. Moreover, stacker-reclaimer schedules dictate when and where stockpiles can be built, in turn affecting the time at which vessels can leave. Therefore, it is sub-optimal to solve the three problems separately.

To fix this issue, one could solve the three problems sequentially per vessel or in an integrated manner. This way, the partial solution of one subproblem affects the other two subproblems. In this thesis, both an integrated and sequential planning approach is presented. The integrated approach takes the form of a mixed integer linear program (MILP). For this problem, however, solving the integrated model (to optimality) is intractable for real-world instances (Bierwirth and Meisel, 2010). Therefore, the exact model is used in small instances to assess the quality of the solutions of sequential approaches: a construction heuristic combined with different local searches.

The rest of the thesis is organised in the following manner. First, we treat the literature on the stockyard planning problem in Section 2. Then, in Section 3 we define the stockyard planning problem. In Section 4 the integrated and sequential methods used in this thesis are treated. The data is explained in Section 5. Then, the results of the exact formulation and heuristics are treated in Section 6, as well as an analysis on stockpile splitting. Finally, we give a conclusion and recommendations for future planners and further research in Section 7.

2 Literature

As discussed, the stockyard planning problem consists of three subproblems: the Berth Allocation Problem (BAP), Yard Allocation Problem (YAP) and the Stacker-reclaimer Scheduling Problem (SRSP). All three subproblems are well-known and studied problems, although they may differ in implementation, which results in different solution techniques. We first discuss the differences in modelling choices for each subproblem. Then we treat the methods by which the SPP is solved in the literature as an effect of these differences. Lastly, we discuss the contribution of this work to the literature.

2.1 Individual subproblems

The Berth Allocation Problem is a well-studied problem that mostly concerns container and dry bulk shipping. It can be divided into three different categories: discrete (Imai et al., 2001), hybrid (Nishimura et al., 2001) and continuous (Park and Kim, 2005). In discrete BAPs the quay consists of a discrete number of berths at which a vessel can berth. In the hybrid instance, vessels are assigned to discrete berths, but are also allowed to exceed it in case they are bigger than the space of the berth. In the continuous case, the vessel can berth at any position of the quay. Naturally, from a discrete to continuous model, the problem becomes harder and computational time increases as found by Mauri et al. (2016). Moreover, the berthing times can be discrete (Meisel and Bierwirth, 2009) or continuous (Cordeau et al., 2005). A recommended survey on BAPs is the one from Bierwirth and Meisel (2015).

The Yard Allocation Problem can also be categorized into a continuous and discrete time and space interval. Most of the time though, the YAP is used with a discrete space as otherwise the solution space becomes infeasibly large. Moreover, YAPs either have the possibility to increase the size of a yard item (Chen et al., 2002) or to decrease the size (Sun et al., 2020). Some dry bulk yard problems also allow for splitting the stockpiles into two when one does not fit (Lipovetzky et al., 2014) and requiring buffer zones between different stockpiles to ensure raw material is not mixed (Kim et al., 2009).

The Stacker-reclaimer Scheduling Problem (SRSP) is the last subproblem that sometimes is excluded from the stockyard planning problem. It has discrete or continuous time intervals, depending on the choice of interval in the Yard Allocation Problem. Each stacker-reclaimer may have availability times which are initial ready times, initial positions or account for the last position of the machine (Bierwirth and Meisel, 2015). Lastly, the SRSP may have a non-crossing constraint, indicating that two cranes on the same track may not cross each other (Unsal and Oguz, 2019).

2.2 Sequential and integrated stockyard problems

Now the differences in implementation details are known, we discuss the literature that solves the stockyard planning problem. All the upcoming literature on the SPP include objective functions minimising vessel delay. First we treat literature using sequential approaches and then we treat the integrated approaches. Sequential approaches usually imply heuristic methods, but they may also be used in combination with exact formulations.

The most well-known stockyard paper using a sequential approach is the one from Boland et al. (2012). Here, a greedy construction heuristic is used to schedule incoming stockpiles from trains towards incoming vessels in order of a vessel estimated time of arrival (ETA) and 'time to start loading' (TSL), which is updated based on the state of the system. They use a discrete berth allocation and a continuous stockpile assignment. Furthermore, time-space candidate pairs are used in an integer program to determine the best time and position for

the stockpile. They conclude that their TSL based heuristic significantly outperforms its ETA-ordered counterpart, although TSL takes more computation time.

A similar work is the one from Babu et al. (2015), where again an ETA and, similar to TSL, an 'ETA-end of reclaiming time' (ETA-ERT) combination is used in a greedy construction heuristic. They use a discrete berth allocation and a continuous stockyard where stockpiles can be cleared strategically to make room for new incoming vessel stockpiles. Their algorithms achieve better results than the actual mean delay of the port. Also, they conclude that with the right settings of the ETA-ERT method, the algorithm outperforms the ETA construction heuristic.

Singh et al. (2012) combine sequential and integrated methods. They also use a construction heuristic based on ETA. After the heuristic though, the order in which vessels are inserted into the heuristic is adjusted by both a genetic algorithm (GA) and squeaky wheel optimisation (SWO). As SWO consistently outperforms GA, this is used to then improve the solution by a fix and optimize procedure. In this last step, an exact model improves the solution of the SWO by fixing certain set of vessels and improving the remaining vessels. They conclude that the fix and optimize procedure is able to improve the solutions of the SWO, although the last step takes much running time. Therefore, they find that the SWO is most feasible in practice.

Continuing with the integrated approaches, Belov et al. (2014) use an exact model to place stockpiles into continuous pads including non-crossing stacker-reclaimer movement. After obtaining a feasible solution, a fix and optimize procedure is performed on sets of vessels that can be sorted in spatial and time based groups. A varying visibility horizon was tested on the instances to enhance their evaluation of the performance. In this horizon, only a selected number of vessels can be seen by the solution method. Their results indicated that the fix and optimize procedure performed similar to a greedy approach.

Xin et al. (2018) also use a mixed integer linear program to solve a case with a limited visibility horizon from berth to train. Their situation contains one vessel berth, stacker-reclaimer and train loader and they partition the stockyard space into multiple stack rows to reduce the solution space. The dynamic problem is solved using an exact model in a static way and a predictive way using Monte Carlo simulation. The static approach resolves the problem for each time interval within the prediction horizon, where the other predicts the future state of the system by its current actions. They conclude that the predictive methods yielded significantly better results.

Lastly, Unsal and Oguz (2019) propose to solve the three subproblems in an integrated way by Benders decomposition. Using a mixed integer linear program as master problem for the vessel and stockpile assignment and a constraint program for the reclaimer schedule as the subproblem, they are able to solve the problem in relatively short time. Their formulation is relatively rich, although it contains the assumption that stockpiles are stacked instantly. This makes the problem of scheduling stockpiles towards the empty vessels considerably less hard.

2.3 Contribution to the literature

As the work in stockyard planning is not as extensive as for other problems, we try to contribute to the research at hand, which we summarise now. First, to the best of our knowledge, this thesis is the first to introduce partial reclaiming of stockpiles in this stockyard setting. Partial reclaiming means that a pile is not reclaimed fully, but at different points in the schedule, making the size of the pile dynamic in time. To incorporate the partial reclaiming, we introduce the concept of a sub-stockpile as explained in Section 4.1. We also adapt the exact formulation and the heuristic approaches for sub-stockpiles. Methods including these feature are compelling to planners as they are much more applicable for real-world use.

Secondly, we notice from the literature that the work on integrated approaches is very separated from the work on heuristics. In the case of Boland et al. (2012) and Babu et al. (2015)

an exact formulation is given, but it is never used as they argue that the problem demands infeasible computation time. As other stockyard papers and others in similar fields such as Meisel and Bierwirth (2009) conclude, the SPP is indeed so comprehensive that integrated techniques are likely to be infeasible to use in real-life scenarios. Still, exact formulations might serve well to assess the quality of their sequential counterparts. We will fill this gap by using an exact formulation to evaluate the heuristics performance based on the exact results in small instances.

Thirdly, in the work on sequential heuristics, we notice the detail in strategy in which stockpiles are assigned within the stockyard. The efforts to then improve the solutions of the construction heuristic, however, are marginal. The only follow-up methods that are used successfully are TSL and ERT by Boland et al. (2012) and Babu et al. (2015) and a SWO by Singh et al. (2012). Also, no work has compared the competitiveness of any other follow-up method within the same context. A gap in the literature that this thesis therefore tries to fill is to develop more follow-up techniques to improve the initial solutions. The goal is then to compare their competitiveness within the same context to discover which follow-up method works best for the SPP.

Lastly, the literature also neglects why some methods seem to work better. Therefore, we also focus on what properties of the problem challenge these heuristics. By analysing the results of the heuristics, we can potentially find strengths and weaknesses in the created follow-up methods. With such an analysis, one could be be better equipped to improve the heuristics weaknesses for future research.

3 Problem description

We now further define the stockyard planning problem at hand. First, the concepts of vessel and stockpile schedules are explained in Section 3.1. Then, the arrival interval and cost structure of vessel are treated in Section 3.2. The goal in the stockyard planning problem is given in Section 3.3 by giving an overview of the problem. Lastly, Section 3.4 discusses the splitting of stockpiles to more effectively make use of the stockyard.

3.1 A vessel and stockyard schedule

First, we define how we create a vessel schedule. As the relevant case concerns a small quay and incoming vessels do not vary in size too much, we choose to use a discrete berth assignment schedule. Moreover, this allows for more focus and computing time in the stockyard schedule which is believed to be the largest bottleneck. An example of a vessel schedule can be seen in Figure 2 and consists of eight vessels. The vertical axis represents which berth a vessel is assigned to and the horizontal axis stands for the time. The width of each rectangle thus represents the duration the vessel is berthed at the quay. Moreover, the height of each rectangle is the maximum allowed vessel size in the specified berth. As can be seen in Figure 2, rectangles at berth 3 and berth 4 have a smaller height, which means these berths only accommodate vessels of a smaller size. As a berth can only serve one vessel at a time, the rectangles in this schedule may not overlap.

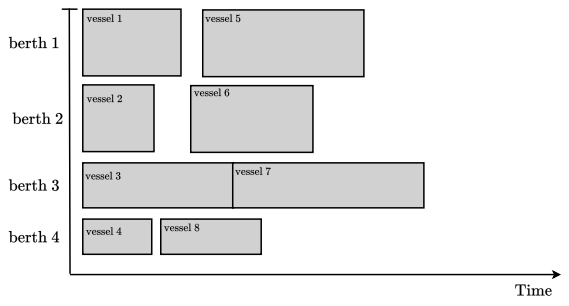


Figure 2: Space-time diagram of vessels in a discretely divided quay.

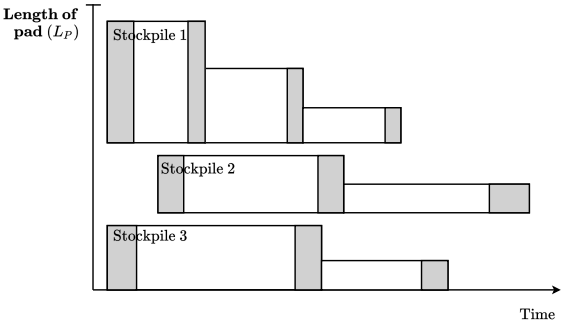


Figure 3: Space-time diagram of stockpiles.

When a vessel has berthed, we have to schedule the stockpiles from the vessel. For this schedule, we choose to use space-time diagrams with a continuous space allocation. The reason for this is that it allows to us to use the stockyard space as much as possible. Moreover, the stockyard pads make use of freely rolling stacker-reclaimers, making it possible to store stockpiles at any location. Figure 3 shows such a space-time diagram. The vertical axis represents the length of a pad (the pad width is always fully utilized) and the horizontal axis stands for the time. The height of a stockpile rectangle represents the space the stockpile takes within the pad. The width of the rectangle is the duration the stockpile is assigned to the space in the pad. In general, each stockpile is fully reclaimed in two or three times, as can be seen by the two or three rectangles together representing the size of the stockpile throughout its life.

Moreover, each stockpile has a stacking and reclaiming time shown as the grey element of the rectangle on left and right, respectively. The left element represents the duration of a stockpile being built. The right element represents the duration the stockpile is being reclaimed after a blast furnace requests the stockpile. For these two periods, at least one stacker-reclaimer has to be available. The rectangles of the stockpiles may not overlap in space and time, as coal of different types may not be mixed. Moreover, there needs to be buffer space between each stockpile to ensure they are not mixed, as shown in the figure. Every pad has a space-time

diagram, together representing the schedule of the complete stockyard.

3.2 Vessel arrival and cost structure

When each vessel could arrive at its ETA, the scheduling in space-time diagrams would be a straightforward process. During the scheduling though, planners often find a case where a vessel might not fit in due to the fact that other vessels occupy the port or that there is no space in the stockyard. In such a case, vessels are allowed to speed up or wait for capacity to free up in the port or stockyard. To ensure that vessels cannot speed up over their speed limit or wait indefinitely, a vessel is given an arrival time interval during which the vessel is able to berth. This time interval is shown as the grey bar beneath the axis in Figure 4 from vessel $v \in V$ earliest starting time EST_v by speeding up to its limit to its latest arrival time LAT_v by waiting. The EST_v is thus the earliest time a vessel can arrive by speeding up and the LAT_v is the latest a vessel can arrive by slowing down or waiting. When the vessel does not speed up or slow down, it arrives at its estimated time of arrival, ETA_v. The time within the interval is the only opportunity the vessel can berth. So when it is not possible to berth within this interval, the vessel is infeasible and incurs cancellation cost c_v^C. In terms of costs, speeding up incurs a cost c_v^{SP} for vessel $v \in V$ per hour it arrives earlier. Lastly, each vessel has a latest finishing time LFT_v after which a demurrage cost c_v^D is incurred per hour the vessel leaves after its LFT. The resulting cost structure can also be seen in Figure 4.

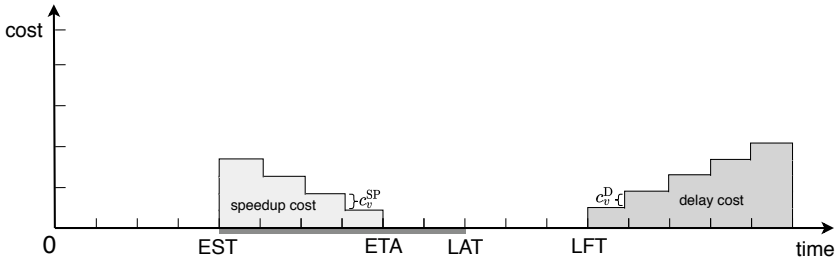


Figure 4: Vessel arrival interval as shown as the grey bar under the axis and cost structure.

3.3 Problem overview

Now the space-time diagrams and cost structure are treated, we continue with the main challenge of the SPP. We first explain when vessel demurrage occurs and we illustrate this with an example. Then, we are able to formulate the SPP problem.

When creating a vessel and stockpile schedule, the ETA of vessels is a given. The stockyard, however, may not have room to accommodate the incoming stockpiles as it has limited capacity to store stockpiles. This results in vessels having to wait for space to become available in the stockyard, leading to demurrage cost. Due to high opportunity costs of vessels, we want to minimise this demurrage. Demurrage cost can be avoided when the stockyard would have room to store the stockpile. Therefore, we have to know what is the cause of the fact that some stockpiles cannot be accommodated in the stockyard. First, it may happen when the stockyard is fully utilized. In other words, every available position in the stockyard is used. Nevertheless, vessel demurrage is mainly caused by a bad placement strategy. A bad placement strategy is a set of decision rules that cause stockpiles to leave undesirably small spaces in both time and space, in which future stockpiles are unable to fit. This effectively results in less usable capacity within the space-time diagram as small spaces in between stockpiles cannot be used for new stockpiles. In this way, such spaces act as unusable capacity.

To illustrate the problem, suppose in Figure 5 we only have this coal pad in the stockyard and stockpile 4 is brought in by vessel 4. Each vessel brings one stockpile corresponding to its vessel number. Vessel 4 has the same ETA as the vessels of stockpiles 1 and 3. Yet, stockpile

4 is just too long to fit at the large cross $\times$ after reclaiming of stockpile 3, so the stockpile can only be scheduled after stockpile 1 is reclaimed. This results in the fact that the vessel has to wait and finishes unloading at e_4 after its latest finishing time LFT, leading to demurrage costs. When stockpiles 1 and 2 had been placed higher in the stockyard, the stockyard would have had more usable capacity at the bottom. In this way, stockpile 4 could have been inserted after stockpile 3 at cross $\times$ at an earlier time. This would have led to less demurrage costs, as in this case vessel 4 departs earlier.

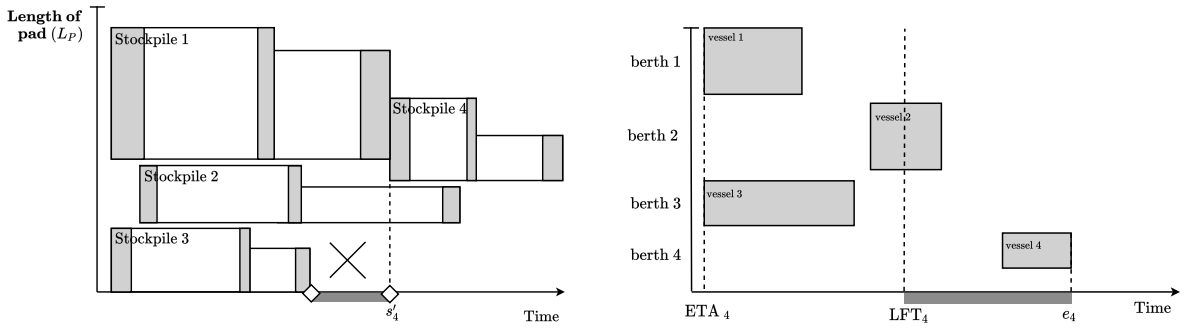


Figure 5: Example of the challenge in the stockyard planning problem.

The example shows that due to the limited stockyard capacity and due to a bad placement strategy, we have to shift a vessel later in time leading to unnecessary demurrage costs. Therefore, the main challenge in the SPP is to optimise the placement strategy of the stockpiles. Approaches to improve this strategy will be treated in Section 4. Also, splitting a stockpile when there is no immediate fit will be discussed in Section 3.4 as an extension to these methods.

The stockyard planning problem can now be formulated as follows. The goal in creating a stockyard schedule is to minimise vessel speed-up, demurrage and cancellation cost, such that the schedule contains non-overlapping schedules for vessels and stockpiles. During the process of stacking and reclaiming there should always be a sufficient number of available stacker-reclaimers and there should be sufficient space in between the stockpiles.

Lastly and importantly, we note that no costs are given to any late reclaiming of stockpiles, which results in late orders to the production process. As our efforts are more focused on the overall fit of the stockyard, we disregard this problem element. This is not to say that this element is not important in a planner's schedule.

3.4 Splitting stockpiles

Lastly, to be able to fit larger stockpiles within a full stockyard, we now introduce the notion of splitting stockpiles into two. By splitting a stockpile, the available capacity in the stockyard could be used more effectively. Nevertheless, we should be careful to not split all stockpiles as this would ask for more stockyard capacity. First, each stockpile will need a buffer space to separate it from other stockpiles. Second and most importantly, a split stockpile collectively needs more space to be placed than it needed without being split. This is explained in the following section.

As the shape of the piles does not allow for simply splitting the length of the stockpile, we may only split in terms of stockpile volume. Therefore, let us first define how the stockpile volume is defined and how the length is computed for a given volume.

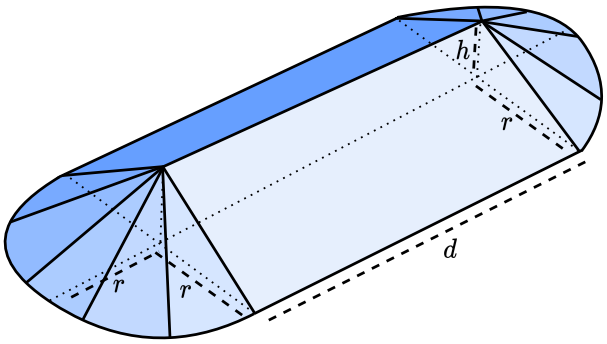


Figure 6: A longitudinal stockpile with two half cone sides and a triangular prism in the middle.

The stockpile, as can be seen in Figure 6, is called a longitudinal stockpile and contains two half cones at the sides and a triangular prism in the middle. This shape is often found in longitudinal domes covering stockpiles that limit the width a stockpile can take. Each cone has width r and height h as does the prism with length d. The pile has a volume approximated by

$$V = hrd + \frac{\pi}{3}r^2h. \tag{1}$$

To eliminate height h let us first look at what is called the angle of repose. The angle of repose α for coal piles is the angle at which the pile is stable and lays between 24° and 30° depending on the type of coal. As this is given we can express the height as $h = \tan(\alpha)r$, so the volume can be re-written as

$$V = r^2\tan(\alpha)d + \frac{\pi}{3}r^3\tan(\alpha). \tag{2}$$

As we want to know the length $l_i = 2r + d$ of stockpile $i \in S$, the width r and prism length d need to be computed. For width r, we first assume the stockpile $i \in S$ is a cone with $d = 0$ and volume $V_i = \frac{\pi}{3}r^3\tan(\alpha)$. Then by using equation (2) we can compute the width r:

$$\hat{r} = \sqrt[3]{\frac{3V_i}{\pi\tan(\alpha)}}. \tag{3}$$

If width $2\hat{r}$ is smaller than the pad width w_p, the pile is indeed a cone, $d = 0$ and the length of the stockpile equals its width $l_i = 2\hat{r}$. When width $2\hat{r}$ is greater than the pad width w_p, the stockpile is longitudinal, $d > 0$ and $\hat{r} = \frac{w_p}{2}$. What remains is computing prism length d. The volume of this prism V_P equals the volume of the whole pile V_i minus the volume of the two half cones $V_P = V_i - \frac{\pi}{3}r^3\tan(\alpha) = V_i - \frac{\pi}{3}\frac{w_p^3}{2^3}\tan(\alpha) = V_i - \frac{\pi}{24}w_p^3\tan(\alpha)$, where r has been replaced by known w_p. As we concluded in (1) and (2), the volume of the prism also can be written as $V_P = r^2\tan(\alpha)d = \frac{w_p^2}{4}\tan(\alpha)d$, where again r has been replaced by w_p. Therefore, the length of the prism can be computed by

$$\hat{d} = \frac{V_i - \frac{\pi}{24}w_p^3\tan(\alpha)}{\frac{w_p^2}{4}\tan(\alpha)}, \tag{4}$$

in which case the length of stockpile $i \in S$ equals $l_i = w_p + \hat{d}$.

When we split a stockpile, we do not split the length, but we split the volume V_i of the stockpile $i \in S$. With the previous computations, we are now able to compute the new length l_i of stockpile $i \in S$ for the volume divided by two. In this way, the stockpile can still be fit in the stockyard. Nevertheless, splitting a stockpile also results into relatively more space being used within the space-time diagram. To illustrate this result, Figure 7 shows the length each

stockpile would take up in the pad from a birds-eye-view when we would lay them side by side. Note that this is not comparable to the space-time diagram. What can be seen is that each time the stockpile is split into one more stockpile, the space they take up collectively increases. This is due to the fact that the two cones at the side can hold less coal than the triangular prism, due to the nature of its shape. What can be shown, is that the total length of these n split piles grows linearly. This is due the fact that the total length increase $\Delta S_L(n)$ as a function of n splits follows

$$\Delta S_L(n) = (1 - \frac{\pi}{6}) w_p (n - 1). \tag{5}$$

Therefore, it does not take a large amount of extra space to split the piles and it is reasonable to split the pile into two stockpiles, as the summed length increases by the $(1 - \frac{\pi}{6})(2 - 1) \approx 0.476$ times the pad width w_p. The proof for this relation can be found in Appendix A.1.

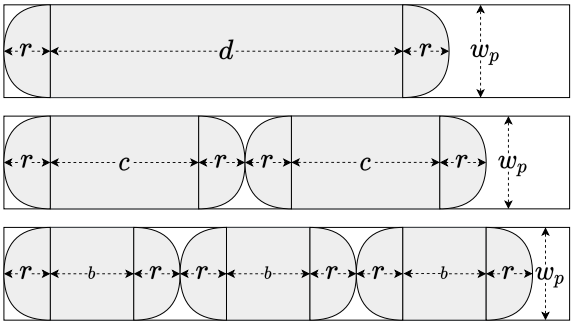


Figure 7: Top view example of a stockpile with prism length d split into two and three with resulting prism lengths c and b and half-cone lengths of r.

4 Methodology

In this section, the mathematical formulation of the stockyard planning problem is given and the heuristic techniques are presented. In Section 4.1, the problem parameters and decision variables are given. Then, an exact model is formulated in Section 4.2, which is enhanced by valid inequalities presented in Section 4.2.1. Section 4.3 develops the construction heuristic, which will form the basis of the heuristic methodology. Follow-up techniques are discussed in Section 4.4, in the form of a squeaky wheel optimisation (Section 4.4.1), a genetic algorithm (Section 4.4.2) and an ant colony optimisation procedure (Section 4.4.3).

4.1 Notation

Relevant parameters are now given for the mathematical formulation. Each vessel $v \in V$ has an earliest starting time EST_v, estimated time of arrival ETA_v, latest arrival time LAT_v and a latest finishing time LFT_v. Moreover, each vessel has a length l_v, a speed-up cost c_v^{SP}, demurrage cost c_v^D for leaving after its latest finishing time LFT_v and a cancellation cost c_v^C for infeasibility by not being able to start before LAT_v. The speedup and demurrage cost are incurred per hour, while the cancellation cost is a fixed penalty. Then, each vessel v has stockpiles $s \in S_v$ and what we will coin as 'sub-stockpiles' $i \in I_s$. These sub-stockpiles are the rectangles in Figure 3 and together represent the stockpile throughout its life. The term 'sub-stockpile' will only be used in the context of the exact model. Each sub-stockpile $i \in I_s$ has a duration to unload d_i^U, transport d_i^T, stack d_i^S and reclaim d_i^T the stockpile. A sub-stockpile $i \in I_s$ has a time t_i when it is requested to be partially or completely reclaimed for the blast furnace. Moreover, each sub-stockpile $i \in I_s$ has a volume V_i, length l_i in the stockyard as the pad width is always fully utilized and a number n_i. This number equals one when it represents the complete stockpile and each following number represents the stockpile rectangle in Figure 3 throughout its life. Next, each pad $p \in P$ has a width w_p and length L_p. Finally, the length of the buffer zone between each stockpile is l_{buffer} and time horizon of the vessel and stockpile schedule is T days.

Table 1: An overview of the input data and decision variables.

Input data		**Decision variables**	
V	set of vessels	ΔETA_v	speed up in hours of vessel $v \in V$
B	set of berths	ΔLFT_v	demurrage in hours of vessel $v \in V$
S, I	set of stockpiles, set of sub-stockpiles	u_v	1 when vessel v is cancelled, zero o.w.
S_v, I_s	stockpiles, sub-stockpiles carried by vessel $v \in V$, stockpile $s \in S_v$	s_v	start/mooring time of vessel $v \in V$
P	set of pads	e_v	end/departing time of vessel $v \in V$
SR_p	stackers-reclaimers used by pad $p \in P$	Z_{bv}	1 when vessel v is moored at berth b, zero o.w.
EST_v	earliest starting time of vessel $v \in V$	X_{bvw}	1 when vessel w is moored later than vessel v at berth b, zero o.w.
ETA_v	estimated time of arrival of vessel $v \in V$	Z_{pi}	1 when stockpile i is placed in pad p, zero o.w.
LAT_v	latest arrival time of vessel $v \in V$	X_{pij}	1 when stockpile j is placed later than stockpile i at pad p, zero o.w.
LFT_v	latest finishing time of vessel $v \in V$	Y_{pij}	1 when stockpile j is placed higher than stockpile i at pad p, zero o.w.
l_v	length of vessel $v \in V$	s_i	stacking start time of stockpile $i \in I$
c_v^{SP}, c_v^D, c_v^C	speed-up, demurrage and cancellation cost of vessel $v \in V$	e_i	end time of reclaiming stockpile $i \in I$
d_i^U	unloading duration of stockpile $i \in I$ in minutes	b_i	position of stockpile $i \in I$
d_i^T	transport duration to stockyard of stockpile $i \in I$ in minutes		
d_i^S	stacking duration of stockpile $i \in I$ in minutes		
d_i^R	reclaiming duration of stockpile $i \in I$ in minutes		
t_i	time at which stockpile $i \in I$ is requested for the blast furnace		
V_i, l_i	volume and length of stockpile $i \in I$		
w_p, L_p	width and length of pad $p \in P$		
l_{buffer}	Length of the buffer zone between each stockpile		
T	Time horizon of the schedule		
n_i	The number of the sub-stockpile $i \in I_s$ within stockpile $s \in S$		

Given these parameters, the decision variables can now be defined. The amount of time units that the vessel is sped up is ΔETA_v, the time units after the vessel is expected to depart is ΔLFT_v and u_v equals 1 when a vessel is cancelled, which all determine the objective value. These are influenced by the decision when to berth s_v and the departure time e_v of vessel $v \in V$. For the exact model we formulate Z_{bv} that equals 1 when vessel v is moored at berth b and zero

otherwise. Variable X_{bvw} equals 1 when vessel w is moored later than vessel v at berth b and zero otherwise. The same variables apply for the sub-stockpiles, but with the location variables added as Y_{pij}, which equals 1 when sub-stockpile j is placed higher than sub-stockpile i at pad p and is zero otherwise. Lastly, each sub-stockpile $i \in I$ has a start time s_i, and ending time e_i and a position in the pad b_i.

4.2 Stockyard planning model

Using the parameters and decisions variables as defined in Section 4.1, we present the following integrated SPP model including the BAP and YAP. As the model is computationally expensive to solve as a result of the abundance of binary variables and big-M type constraints, the model is only used on small instances. The formulation is therefore not applicable for real-world instances. Still, it gives a comprehensive overview of the problem and most importantly, we are able to check the quality of the heuristic based approaches in small use cases. Solving the model and comparing exact and heuristic solutions for smaller instances namely gives us some indication on how well the heuristics perform in larger instances. The model excludes the stacker-reclaimer scheduling problem.

$$\min \quad \sum_{v \in V} (c_v^{SP} \Delta\text{ETA}_v + c_v^D \Delta\text{LFT}_v + c_v^C u_v) \tag{6}$$

$$\text{s.t.} \quad \Delta\text{ETA}_v \geq \text{ETA}_v - s_v \qquad \forall v \in V \tag{7}$$

$$\Delta\text{LFT}_v \geq e_v - \text{LFT}_v \qquad \forall v \in V \tag{8}$$

$$s_v \geq \text{EST}_v \qquad \forall v \in V \tag{9}$$

$$s_v \leq \text{LAT}_v \qquad \forall v \in V \tag{10}$$

$$e_v \geq s_i - d_i^T \qquad \forall v \in V, s \in S_v, i \in I_s : n_i = 1 \tag{11}$$

$$\sum_{b \in B} Z_{bv} = 1 - u_v \qquad \forall b \in B, v \in V : l_v \leq L_b \tag{12}$$

$$s_w + T(1 - X_{bvw}) \geq e_v \qquad \forall b \in B, v, w \in V : v \neq w \tag{13}$$

$$X_{bvw} + X_{bwv} \leq \frac{Z_{bv} + Z_{bw}}{2} \qquad \forall b \in B, v, w \in V : v \neq w \tag{14}$$

$$X_{bvw} + X_{bwv} \geq Z_{bv} + Z_{bw} - 1 \qquad \forall b \in B, v, w \in V : v \neq w \tag{15}$$

$$s_i \geq s_v + d_i^U + d_i^T \qquad \forall s \in S, i \in I_s : n_i = 1 \tag{16}$$

$$e_i \geq t_i \qquad \forall s \in S, i \in I_s \tag{17}$$

$$e_i \geq s_i + d_i^S + d_i^R \qquad \forall s \in S, i \in I_s : n_i = 1 \tag{18}$$

$$e_i \geq s_i + d_i^R \qquad \forall s \in S, i \in I_s : n_i > 1 \tag{19}$$

$$e_i \leq T \qquad \forall s \in S, i \in I_s : n_i = |I_s| \tag{20}$$

$$\sum_{p \in P} Z_{pi} = \sum_{b \in B} Z_{bv} \qquad \forall s \in S, i \in I_s : n_i = 1 \tag{21}$$

$$b_i + l_i - L_p(1 - Z_{pi}) \leq L_p \qquad \forall p \in P, s \in S, i \in I_s : n_i = 1 \tag{22}$$

$$b_j + L_p(1 - Y_{pij}) \geq b_i + l_i + l_{\text{buffer}} \qquad \forall p \in P, s \in S, i, j \in I_s : i \neq j \tag{23}$$

$$s_j + T(1 - X_{pij}) \geq e_i \qquad \forall p \in P, s \in S, i, j \in I_s : i \neq j \tag{24}$$

$$X_{pij} + X_{pji} + Y_{pij} + Y_{pji} \leq \frac{Z_{pi} + Z_{pj}}{2} \qquad \forall p \in P, s \in S, i, j \in I_s : i \neq j \tag{25}$$

$$X_{pij} + X_{pji} + Y_{pij} + Y_{pji} \geq Z_{pi} + Z_{pj} - 1 \qquad \forall p \in P, s \in S, i, j \in I_s : i \neq j \tag{26}$$

$$b_j = b_i \qquad \forall s \in S, i \in I_s : n_j = n_i + 1 \tag{27}$$

$$s_j = e_i \qquad \forall s \in S, i \in I_s : n_j = n_i + 1 \tag{28}$$

$$Z_{pj} = Z_{pi} \qquad \forall s \in S, i \in I_s : n_j = n_i + 1 \tag{29}$$

$$Z_{bv}, X_{bvw}, Z_{pi}, X_{pij}, Y_{pij} \in \{0,1\} \qquad \forall b \in B, v, w \in V, \forall p \in P, i, j \in S \tag{30}$$

$$\Delta\text{ETA}_v, \Delta\text{LFT}_v, s_v, e_v, s_i, e_i, b_i \geq 0 \qquad \forall v \in V, s \in S_v, i \in I_s \tag{31}$$

Here, the objective function (6) minimises the total speed-up, demurrage and cancellation cost over all vessels. Constraints (7) and (8) define the speed-up and demurrage duration respectively. Constraints (9) and (10) simply ensures that a vessel berths between its earliest starting and its latest starting time. Constraints (11) indicate that the end time of a vessel is later than the start time of any of its stockpile minus the transport time to the pad. Constraints (12) states that a vessel can moor on at most one berth and allows for vessels not to berth when u_v equals 1. Moreover, vessels with a length exceeding the length of the berth are excluded from the berth in the same constraint set. Constraints (13) indicates that the start time of vessel w should be later than the end time of vessel v, when vessel w is scheduled later than vessel v within the same berth b. Constraints (14) and (15) link the decision variables Z_{bv} and X_{bvw}. Moreover, they decide the order between two vessels v and w within the same berth. This is due to the fact that when the berth b is the same, $X_{bvw} + X_{bwv} = 1$ must hold, indicating that either vessel v should be moored before w or vice versa. When in the constraints (15) the two vessels have a different berth, the right hand side becomes zero and the order does not matter.

Continuing with the constraints of the stockpiles, constraints (16) are associated with the start time of a sub-stockpile $i \in I$. The start time of the complete sub-stockpile should be later than the mooring time of its vessel plus unloading and transport time. Constraints (17) ensure that the end time of a sub-stockpile is later than its requested time to the blast furnace. Also, in (18) the end time of the complete sub-stockpiles should always be greater than its start time added with the stacking and reclaiming time, as every stockpile cannot skip the stockyard. This also holds for the other sub-stockpiles in (19), but here they do no have to be stacked. Furthermore, sub-stockpiles need to end before the time horizon in constraints (20). Constraints (21) determines that a stockpile can be allocated to exactly one pad whenever the vessel is berthed and not cancelled. Constraints (22) imposes that the stockpile cannot exceed the limit of the pad. Then, constraints (23) indicates that the lowest position of stockpile j should be greater than the highest position stockpile i takes plus a buffer zone l_{buffer} when stockpile j is placed above stockpile i. The constraint set (24) is the stockpile counterpart of constraint set (13) preventing stockpiles from overlapping in time. Furthermore, the constraints in (25)-(26) work the same as in the vessel case, but now for stockpiles locations as well due to the Y_{bij} variable. Both X_{bij} and Y_{bij} variables can be placed in one constraint as stockpiles following each other in time do not have to be placed above or below each other and vice versa.

Then, the sub-stockpiles that originate from the complete stockpile should have the same starting position, the correct start time and be in the same pad as the complete stockpile described in (27)-(29). Lastly, constraints (30) and (31) define the binary and continuous domain of the decision variables.

4.2.1 Valid inequalities

As the goal of the exact formulation is to validate the performance of the heuristics, we are interested in finding the best and lowest integer solutions. An evident and clear-cut approach to achieve the best upper bound is the use of valid inequalities. Valid inequalities are constraints that are violated by fractional solutions while they remain valid for integer solutions. In this way, valid inequalities cut off a fractional part of the polyhedron, potentially reducing the number of cuts that need to be made by the solver and steering the bounds towards integers solutions.

The first valid inequalities we consider concerns the symmetry of two solutions using the two pads. As the transport time to the two pads and the pad lengths are equal, a solution where the first stockpile is inserted into the first pad is equivalent to one where the stockpile is

inserted into the second pad. Therefore, for the first stockpile we can exclude a possibility of the second pad.

$$Z_{i1} = 0 \quad \forall i \in I_s : s = 0 \tag{32}$$

The second inequality is in the form of a cover inequality as described by Gu et al. (1998). We note that there must be some set of stockpiles C that need to be in the stockyard at the same time for which the pad capacity is exceeded by one stockpile. More specifically, let us define C as the set of stockpiles which length jointly exceed the pad capacity including buffer zone l_{buffer} by one of the stockpiles. Moreover, let the duration between latest arrival time (LAT_v) of the vessels of the stockpiles and the request times t_i overlap with each other. This makes sure that even when the vessel would arrive at its latest time, each stockpile $s \in C$ must be in a pad at the same time. As the stockpiles jointly exceed the pad capacity, a maximum of $|C| - 1$ stockpiles may be in the pad at the same time. Therefore, we can define our first cover inequality with cover C as

$$\sum_{i \in C} Z_{pi} \leq |C| - 1. \tag{33}$$

With this inequality the bounds of the problem are stronger, because $\sum_{i \in C} Z_{pi}, \forall p \in P$ cannot be fractional anymore between $|C| - 1$ and $|C|$. To strengthen it as much as possible, all possible combinations of set C are searched for to increase the probability that a given constraint will be violated.

The same cover C can be used to construct a valid inequality for the decision variables Y_{pij}, which is inspired by vessel cover constraint (20) from Correcher et al. (2019). Here, the authors use the same left hand side of their cover constraint on a non-ordered set of variables similar to Y_{pij}. For the right hand side they use $\frac{(|C|^2-|C|)}{2} - 1$ for which we note that the constraint can be strengthened by using a tighter bound. We illustrate this with an example.

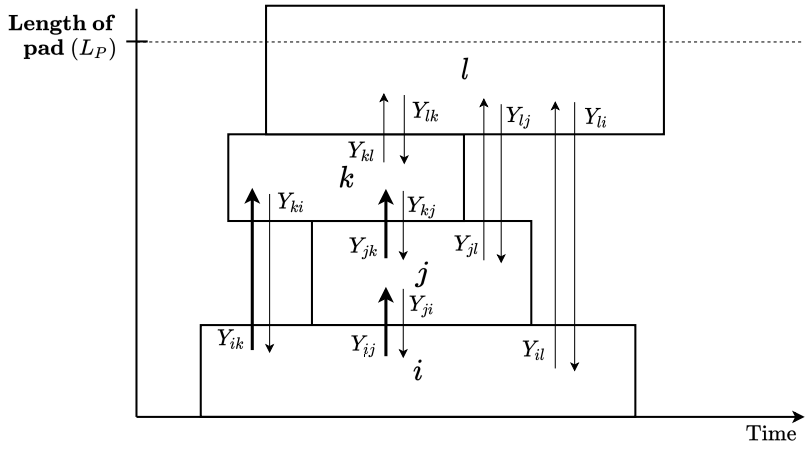


Figure 8: Illustration of the valid inequality (34) with four stockpiles

Consider a maximum of three stockpiles that can fit in a pad, which is the case for all cover sets C in our problem. As can be seen in Figure 8, for these stockpiles a maximum of three Y_{pij} can be activated at the same time as stockpile l does not fit in the pad. For these variables in this situation, the activated ones are indicated as the bold arrows. When we would use the bound of Correcher et al. (2019) we would arrive at $\frac{(|4|^2-|4|)}{2} - 1 = 5$ and we would indeed exclude one variable and with it a fractional part of the polyhedron. We can improve it though by excluding two other variables as well to arrive at a sum of three. In general, we can write this right hand-side as $\frac{(|C|^2-|C|)}{2} - |C| + 1$ to arrive exactly at the maximum number of Y_{pij}

variables that can be activated at the same time, given that $|C| \geq 3$. Therefore, we use the same cover set C to add a valid inequality in the form of

$$\sum_{i \in C} \sum_{j \in C, i \neq j} Y_{pij} \leq \frac{|C|^2 - |C|}{2} - |C| + 1, \quad |C| \geq 3. \tag{34}$$

4.3 Construction heuristic

To be able to provide quick and efficient solutions, we now propose a sequential construction heuristic. The construction heuristic consists of two parts. First we schedule a vessel by the procedure in Figure 9 as explained in Section 4.3.1 and then directly schedule the stockpiles of the vessel with the procedure in Figure 12 in Section 4.3.2. Stockpiles of vessels are scheduled along with each vessel, as the placement of stockpiles has direct influence on when other vessels can berth.

4.3.1 Vessel insertion

Starting with the vessel insertion procedure in Figure 9, we initialize the first berthing time s_v equal to its ETA_v in step (a). For the selected berthing time s_v, we find all free berths in step (b). If there are no free berths, we select a new time berthing time s_v in step (g). Selecting a new berthing time s_v is done using the possible berthing interval $[\mathrm{EST}_v, \ldots, \mathrm{ETA}_v, \ldots, \mathrm{LAT}_v]$ as discussed in Section 3.2. When we pick a new time s_v, we first go back in time to the earliest starting time EST_v of vessel $v \in V$ and then continue with times after the ETA_v until the latest arrival time of vessel $v \in V$ LAT_v. The reason for first speeding up the vessel is that speeding up is less costly than obtaining a demurrage cost.

When a free berth has been found, all the stockpiles on the vessel are inserted into the stockyard in step (c) according to the procedure in discussed in the next section. The stockpiles are inserted in an order that is specified by the vessel $v \in V$. The stockpile procedure returns an end time e_v for the vessel given by the the stacking time of the stockpile with the latest starting time $s_{i'}$ minus the transport time, $e_v = s_{i'} - d^T_{i'}$, which is set in step (d). In the same step, the vessel start time s_v is adjusted by the start time of the first inserted stockpile i'' minus the transportation and unloading time, $s_v = s_{i''} - d^T_{i''} - d^U_{i''}$. In this way, the vessel is scheduled only for unloading stockpiles, which leaves more space for other incoming vessels.

If these assignments are successful and result in a non-overlapping schedule without any vessels on the same berth, we have found a feasible assignment and we save it if it was the best found solution. When the stockpile insertion results in an end time that overlaps with another vessel in time, we pick a new berth b_v for vessel $v \in V$ in step (e). If no berth has been found, we pick another berthing time s_v from the known berth time list in step (g). When selecting a new berthing time s_v, we check for two conditions. First if the list is exhausted, we terminate the procedure. Secondly, we also stop if the next starting time s_v is greater than ETA_v and we have already found a solution. This is done to prevent any unnecessary assignments incurred with delay to speed up the heuristic.

Finally, if there was a successful vessel insertion, the vessel is inserted into berth b_v with start time s_v and end time e_v and we update the state of the system. If no feasible assignment has been found, the vessel is cancelled for cost c^C_v.

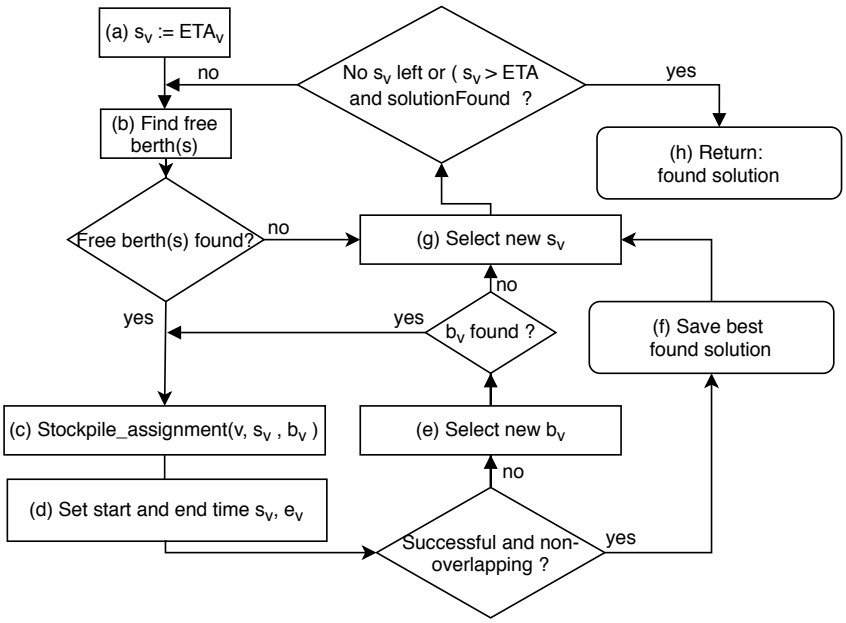


Figure 9: Insertion heuristic for the vessels

4.3.2 Stockpile insertion

As each vessel is inserted through the previous procedure, the stockpile procedure in Figure 12 inserts the stockpiles of its vessel into the stockyard. The procedure is thus repeated for each stockpile $s \in S_v$ when inserting a vessel $v \in V$ in the previous procedure. The logic is the same although here, continuous contact points are considered instead of discrete starting times and berthing locations. Moreover, we terminate the search in a pad as soon as a stockpile location has been found, as no demurrage costs can be evaluated here.

First, we search for a location for stockpile i in the first pad p_i. We search for all contact points (b_{ip}, s_{ip}) with position b_{ip} and start time s_{ip} for stockpile i in pad $p \in P$ in the space-time diagram. These contact points serve as a way to minimise the search space in the diagram, but also to schedule in the stockpiles as compactly as possible.

An example of the generation of contact points can be seen in Figure 10. In this example, for simplicity purposes each stockpile is reclaimed fully in one action. The logic works the same with stockpiles being reclaimed in two or three times. As can be seen in the figure, both scheduled stockpiles get four contact points: upper-left , upper-right, lower-left and lower-right $(1, 2, 3', 4')$. For the upper-left and lower-right corners, the new stockpile always gets connected with the lower-left corner. For the lower-left and upper-right corner, the new stockpile always get connected with its upper-left corner. This is done to maximise the vertical contact with other stockpiles in the space-time diagram, resulting in maximising the left-over available space in the pad. Moreover, for each stockpile's end time we add a contact point at other time-overlapping stockpiles with the same end time of the original stockpile at the top and bottom $(1', 2')$. In this case the top and bottom of stockpile 2 gets a contact point on top with the same end time as stockpile 1. Lastly, we add a contact point on the bottom and top of all stockpiles, $(1'', 2'')$ which are already located at the time at which the stockpile arrives, $s_{v_i} + d_i^U + d_i^T$, with that time. In the figure this is the dashed vertical line and stockpile 2 thus gets two contact points added. The faded diamonds represent infeasible and the circles are feasible contact points.

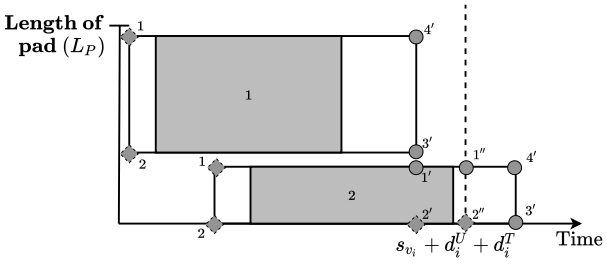


Figure 10: Small example of the contact points

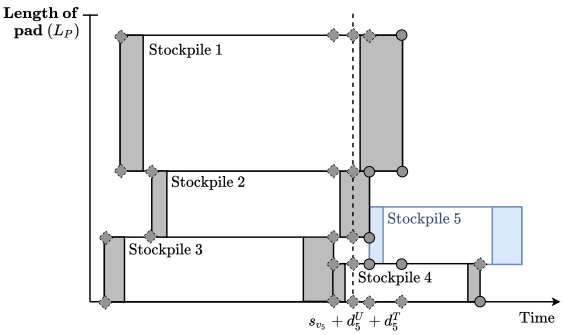


Figure 11: Large example where stockpile 5 is inserted

All these contact points together are sorted based on proximity to the arrival time of the stockpile, $\text{ETA}_{v_i} + d_i^U + d_i^T$. When multiple points have the same time, we distinguish between different priority rules. First, top-left has priority over bottom-left (1 over 2 in Figure 10) as we build up from the bottom. Secondly, for contact points on the right-end and end-times on top and bottom of other stockpile we prioritise the contact points of the other stockpiles, first top then bottom ($1'$ over $2'$) for the same reason as before. The rationale behind first selecting other stockpiles is to maximise the left-over space in the stockyard. Moreover, we select the contact point of a stockpile with the most resulting overlapping contact. For the end-points of the original stockpiles, we prioritise bottom over top ($3'$ over $4'$) as we build from the bottom. Lastly, for the contact points with the arrival time, we again prefer top over bottom ($1''$ over $2''$) and we give priority to the stockpile that gives the most overlapping time contact with the inserting stockpile or the bottom of the stockyard.

In this way, we try to fit in the stockpile such that the vessel has to wait as little as possible. All these contact points not only make sure that a stockpile can be fitted as compactly as possible in both time and space, but they also ensure we minimise the search space. A bigger example where the best feasible contact point is evaluated is shown in Figure 11. In this example, stockpile 5 is inserted at the shown contact point as a result of the priority rules previously discussed.

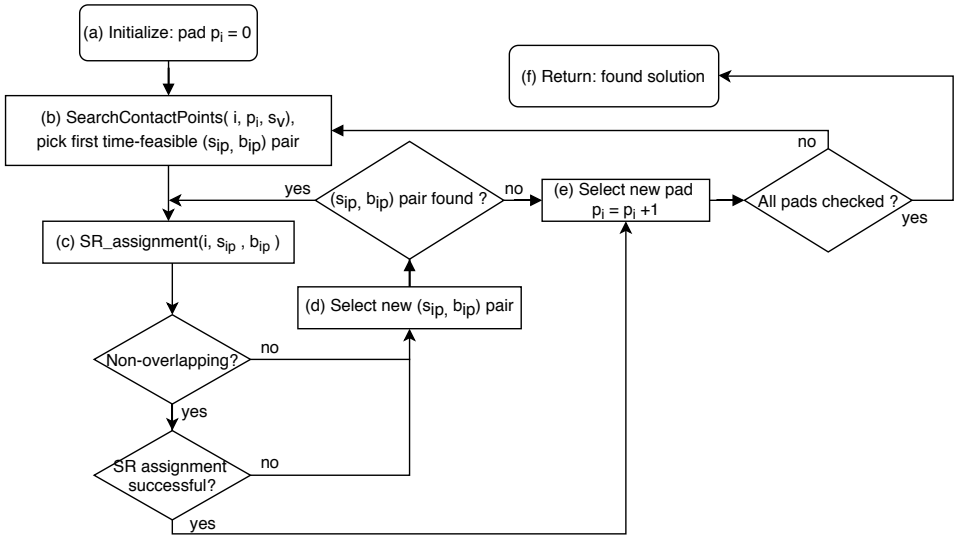


Figure 12: Insertion heuristic for the stockpiles

When the allocation is non-overlapping in both time and space, we continue with the stacker-reclaimer assignment in step (c) in Figure 12. For this, it holds that for the stacking and the

reclaiming duration, one stacker-reclaimer needs to be available. We schedule the stacking as soon as possible while satisfying $s_i \geq s_{v_i} + d_i^U + d_i^T$ (stacking can start after unloading and transport to the stockyard). When a stacking time has been found, we continue with the reclaimer assignment. For this, we look at the time t_i at which the stockpile is needed for an order. This is used to get an end time for the stockpile that satisfies $e_i \geq t_i$ (the end time is after the time of the order) and the fact that the end time should be later than the start time plus any stacking time and reclaiming happening in the mean time. Again, we schedule the available reclaimers as soon as possible.

If the stacker-reclaimer assignment is successful, we succeeded in placing the stockpile and we terminate the procedure for the current pad p_i. In case of an overlap or failed stacker-reclaimer allocation, we pick a new space-time pair (b_{ip}, s_{ip}) in step (d). If a pair is found or when all pairs have already been evaluated, we start the search in a new pad in step (e). If the pads are exhausted we terminate the procedure and we give an end time e_v to vessel $v \in V$. A feasible allocation can always be found as it is always possible to place a stockpile later in time in the stockyard.

4.4 Local search - vessel order

As the construction heuristic inserts each vessel one at a time, the algorithm needs an order in which the vessels are inserted. The order in which the vessels are inserted uses what is called a vessel priority list x. Particularly, a standard approach to make up the list x would be to order the vessels based on ETA. Still, we might be able to improve the solution by altering the priority list x by moving some vessels up in priority before the execution of the construction heuristic. The idea behind giving some vessels more priority is that they get a better stockpile placement in the stockyard such that vessel demurrage is avoided as explained in Section 3.3. By giving priority to high-cost vessels in the vessel order, the cost of the prioritized vessel can be significantly reduced, while the other vessels moving down in priority list only have a small or no cost increase, leading to lower overall demurrage cost. Note that a higher priority does not mean the vessel arrives earlier in the terminal, but we only insert the vessel before other vessels in the algorithm.

As a vessel list of a few dozen vessels can be ordered in $n!$ number of ways, it is unrealistic to evaluate all possible orderings. And although the construction heuristic might perform quickly, the given computation time will limit the amount of evaluations we can make with the heuristic. Moreover, one can imagine that with vessels arriving in separate time windows, the latest arriving vessel does not have to be inserted first to obtain a good solution. The improvement in cost might already by achieved by giving the vessel more priority over vessels that arrive around the same time. This is due to the fact that their stockpiles are in the stockyard around the same time. Therefore, we refrain from searching for the best insertion order in big solution spaces and we focus on elementary swaps between two vessels. This means we only shift vessels one place up in priority.

The following local searches to improve upon the construction heuristic are presented. A squeaky wheel optimisation in Section 4.4.1 as inspired by Meisel and Bierwirth (2009), a genetic algorithm in Section 4.4.2 and an ant colony optimisation procedure developed to challenge these methods in Section 4.4.3.

4.4.1 Squeaky Wheel Optimisation

A commonly used method in this type of setting is called squeaky wheel optimisation (SWO). The benefit of SWO is that it is able to quickly improve solution orders by looking at individual vessel costs making up the overall solution. The term 'squeaky wheel' comes from the fact that we try to fix individual 'squeaky wheels' within the solution to come up with a better overall solution. To do this, we make use of the fact that the objective can be decomposed into the

costs of each vessel. Therefore, each high-cost vessel can be compared to a squeaky wheel in the analogy. When SWO detects a vessel with high cost, it prioritizes this vessel such that the solution's 'squeakiness' is resolved.

The general algorithm is listed in Algorithm 1. First, the vessel list in order of ETA is evaluated by the construction heuristic to obtain the best starting cost z^*. Then, we intensify the solution by only moving one vessel up in priority by starting with the highest-cost vessel and moving down in vessel cost. We continue the intensification process until no better solution can be found by moving one vessel up in priority. When the intensify method is finished, we diversify the solution until a better starting solution with objective $z < z^*$ has been found to diversify with or if the diversification has not yielded a better starting solution for N_{div} diversifications. Furthermore, before we diversify we always revert to the best found solution order x^*. This is done to best utilise the short given computation time relative to the time the construction heuristic takes. A diversification consists of also moving one vessel up in order. This seems to resemble the intensification step, but the difference here is that the vessel swap is always carried out even with a resulting worse solution objective. When a diversified solution has been accepted, we continue with the intensification and diversification thereafter until the time limit has been reached. During the algorithm, we keep track of the best found vessel order x^*.

Algorithm 1 Squeaky wheel optimisation

```
1:  procedure SqueakyWheelOptimisation(vesselOrder x_ETA)
2:      z* ← constructionHeuristic(x_ETA)
3:      while Time limit not reached do
4:          x ← intensify(x)
5:          while z ≥ z* OR diversificationFailed < N_div do
6:              x ← revert(x*)
7:              x ← diversify (x)
8:              z ← constructionHeuristic(x)
9:          end while
10:     end while
11:     return x*          ▷ During the algorithm, we keep track of the best found order
12: end procedure
```

4.4.2 Genetic Algorithm

The second method is a genetic algorithm (GA). An unknown variant of GA is used by Singh et al. (2012), but they conclude that the GA is ineffective in finding good solutions. Although their problem slightly differs and their specific implementation settings are not presented, it still might be interesting to add the method to our list of comparison. Another reason for our inclusion of GA is its potential to find quick but also very diverse solutions by using cross-over of two solutions and parallel evaluation of multiple solutions in the population.

The idea behind a genetic algorithm is using the concept of 'survival of the fittest' and apply it to optimisation problems by generating a lot of potential solutions in a population. The implementation is described in Algorithm 2. First, we create a population N of $n_{\text{population}}$ random vessel orders by moving one random vessel up in priority from the ETA order. Then, for each individual in the population we evaluate its fitness as the objective value of the construction heuristic. When the whole population N has been evaluated, GA selects the best half of the population to form random pairs of parents. For each parent, we create two offspring solutions by applying cross-over.

Cross-over works by first selecting one parent and taking its order for between 25% and 75% of the vessels and adding it to the child. After the first parent, the remaining vessels that are not yet added to the child's order are added in the order of the second parent. For example,

parent 1 (1,4,3,2,5) and parent 2 (1,5,3,2,4) with a cut between the third and fourth vessel produces offspring as (**1,4,3**,5,2) as the first three vessels in bold are a direct copy of the first parent. The second half is in the order of the second parent. For the second child, the parents get swapped and another random cutoff point is determined.

Lastly, some children are mutated randomly with probability p_{mutate} by moving one random vessel one place to the front. These children are added to the population N and the worst half of the population is removed. For each individual, again the objectives are determined, parents are created and we continue until the time limit has been reached. The best settings in terms of population size and mutation rate are determined by sensitivity analysis.

Algorithm 2 Genetic algorithm

procedure GeneticAlgorithm(`vesselOrder` x)
  Create a child population of size $n_{\text{population}}$ random vessel orders
  while Time limit not reached **do**
    for each `child` c **do**
      $z_c \leftarrow$ `constructionHeuristic`(x_c)
    end for
    Select $\frac{n}{2}$ of best children and form pairs randomly
    for each `pair` **p do**
      $x', x'' \leftarrow$ `formOffSpring(p)` ▷ We create two children as offspring
      Apply mutation with probability p_{mutate}
      $N \leftarrow N \cup \{x', x''\}$
    end for
    Remove half of the worst population
  end while
  return x^* ▷ During the algorithm, we keep track of the best found order
end procedure

4.4.3 Ant Colony Optimisation

Lastly, we propose an ant colony optimisation (ACO) procedure to improve the objectives of the construction heuristic. A benefit of ACO is the parallel evaluation of very diverse solutions just as in the genetic algorithm. A potential useful difference here though is that each ant is vastly more diverse in each iteration than in GA, as the ant builds a random path each iteration. Therefore, it might outperform GA in terms of diverse solutions. The basic idea behind ACO is using pheromone levels to guide the solution into low-cost solution spaces. Pheromones are used by blind ants to communicate to other ants where they have been. Ants smelling paths with high amounts of pheromone have a high probability of following a previously made path, reinforcing the paths that a set of ants collectively follow. The trick in ACO is to only create pheromone levels for low cost paths.

The way we do this for our SPP is described in Algorithm 3. The procedure first creates $|A|$ ants and initializes the pheromone levels evenly. Then, while the time limit has not been reached we randomly perturb the order of the given vessel list x, which will be explained after the algorithm. The perturbation for each ant is based on the pheromone levels τ and we check its objective z_a by using the construction heuristic. When this has been done for all ants $a \in A$, we update the pheromone levels based on the found objectives z. This is done such that low cost orders of vessels obtain a higher pheromone level and high cost orders do not obtain pheromone. Based on these new pheromone levels, we again randomly create orders and we continue until the time limit has been reached. Throughout the algorithm, we keep track of the best performing vessel order x^*.

Algorithm 3 Ant colony optimisation

procedure AntColonyOptimisation(`vesselOrder` x)
  Create $|A|$ ants and initialize the pheromone levels
  while Time limit not reached **do**
    for each `ant` a **do**
      $x'_a \leftarrow$ `perturbOrder`$(x, \boldsymbol{\tau})$ ▷ Randomly change order based on pheromone levels
      $z_a \leftarrow$ `constructionHeuristic`(x'_a)
    end for
    $\boldsymbol{\tau}' \leftarrow$`updatePheromone`$(\boldsymbol{\tau}, \mathbf{z})$ ▷ Change pheromone based on objectives $\mathbf{z}$
  end while
  return x^* ▷ During the algorithm, we keep track of the best found order
end procedure

We will now extend on how the vessel order is perturbed and how the pheromone levels are updated. To follow the notion elementary swaps as discussed in Section 4.4, we define the pheromone levels as τ_{vi} for vessel $v \in V$ and $i \in \{0, 1\}$, where i is 1 when we move vessel v forward one place in priority over the original ETA order and 0 when we do not. The main drawback of this approach is therefore that a vessel can only be moved one place forward or back instead of multiple places in priority as compared to in the SWO and GA. The probability then that vessel $v \in V$ gets priority based on value $i \in \{0, 1\}$ can be defined as

$$p_{vi} = \frac{\tau_{vi}}{\sum_j \tau_{vj}}. \tag{35}$$

The probabilities p_{vi} and pheromone levels τ_{vi} are the same for each ant $a \in A$. When each ant a's random vessel order has been evaluated by the construction heuristic, the ACO updates the pheromone levels such that for both low-cost and often used vessel orders, the pheromone τ_{vi} increases. This is done by using

$$\tau_{vi} = (1 - \rho)\tau_{vi} + \sum_{a \in A(vi)} (Q - z_a)^+. \tag{36}$$

Here, $A(vi)$ is the set of ants that use priority level i for vessel v and z_a is the objective of ant $a \in A$. Then, ρ and Q defined as α times the best found objective throughout the ant colony search $Q = \alpha z^*$ are chosen by experimentation. By defining Q in this way, low-cost solutions obtain higher pheromone, while solutions above Q get no pheromone. Using this equation, part of the previous pheromone evaporates by $(1 - \rho)\tau_{vi}$ and for both low-cost and often used ants that use priority i, τ_{vi} increases. This is due to $(Q - z_a)^+$ increasing when z_a is small and $\sum_{a \in A(vi)}(Q - z_a)^+$ increasing as ant a can more often be found in the set of ants that use priority i for vessel v, $A(vi)$.

5 Data

To test both the exact model and heuristic methods, four sets of instances are generated containing 10, 15, 30 and 75 vessels. The first two sets will exclusively be used for the exact formulation and contain 20 instances. The sets of size 30 and 75 vessels consist of 30 randomly generated instances to ensure the performance of each heuristic method is thoroughly tested. In each instance three vessel classes are used: HandySize, HandyMax and SupraMax, which account for roughly 60%, 30% and 10% of the vessels respectively. The technical specifications of these vessel classes are listed in Table 2. As can be seen, each vessel has a specific length l_v to specify for the size of the accommodating berth and the volume of each carried stockpile V_i, which is uniformly distributed. As the vessel classes differ in capacities, the HandyMax and SupraMax are able to carry more stockpiles $|S_v|$ which is binomially distributed with $p = 0.5$. Moreover, each vessel class has a different cost for speeding up c_v^{SP} and for delay c_v^{D} per hour. We note that the larger the vessel, the greater its opportunity costs and the greater the cost of speeding up due to the mass of each vessel class. The cancellation cost for a vessel is 0.1 times the total volume in tonnes on board plus 10,000 ($c_v^C = 0.1 \cdot \sum_{i \in S_v} V_i + 10,000$) to have vessels with more cargo have a higher penalty. We refrain from unitizing costs as we focus on the relative costs and to decrease the size of the found objectives.

For each instance, every two days one vessel with an ETA distributed uniformly over the day is generated. Moreover, the earliest starting time is exactly one day before a vessel's ETA. The latest arrival time LAT is always three days after a vessel's ETA and its latest finishing time LFT five days after the ETA. Each 'sub-stockpile' has a reclaim duration of two times the volume in thousands of tonnes in hours ($d_i^R \propto 2V_i$) and a stacking duration as the volume in thousands of tonnes in hours ($d_i^S \propto V_i$). Here, the stacking duration is shorter as stacking takes less time than unloading and reclaiming. Each stockpile is either a two-time reclaimed stockpile or reclaimed in three times with probability $p = 0.5$. Moreover, the request time t_i for the final and last reclaim of a stockpile is uniformly distributed between the vessel's LFT and 20 days after the LFT. The request times in between are uniformly distributed and $U(0.4, 0.6)$ times the sampled last request time for a two-time reclaimed stockpile and $U(0.2, 0.3)$ and then $U(0.4, 0.6)$ times the sampled final request time for a three-time reclaimed stockpile.

Finally, there is one small, one medium and two large berths. The stockyard consists of two pads with a width $w_p = 50$m and a length $L_p = 700$m and two stacker-reclaimers $n_p^{\text{SR}} = 2$ for each pads $p \in P$. The transport time d_i^T equals five minutes, the angle of repose $\alpha = 28°$ and for the exact model the time horizon T consists of 60 days for the instances of 10 and 15 vessels.

Table 2: Technical specifications of the different vessel classes. For each vessel class, a length l_v, stockpile volume V_i, number of stockpiles $|S_v|$, speed-up c_v^{SP} and demurrage costs c_v^{D} are given.

Class	l_v	V_i (x1000 tonnes)	$\lvert S_v \rvert$	c_v^{SP}	c_v^{D}
HandySize	small	U[15,25]	1	10	20
HandyMax	medium	U[15,25]	B[1,2]	20	40
SupraMax	large	U[15,25]	B[2,3]	30	50

6 Results

In this section, we present the results from the proposed exact model and the heuristic approaches. First, the settings of the computational experiments are listed in Section 6.1. Then, the valid inequalities are tested on the exact model in Section 6.2. The exact model is used to assess the performance of the heuristic approaches in Section 6.3. In Section 6.4, the competitiveness of each local search in vessel order is tested against the benchmark of first come first serve. To verify what problem properties challenge the heuristic's objectives, we evaluate the instances in Section 6.5. With this analysis we search for methods that might be a best practice in certain conditions in Section 6.6. Finally, in Section 6.7 we extend the problem by evaluating when it is optimal to split stockpiles when they do not fit in the stockyard.

6.1 Settings

For all results, the exact model is implemented in `CPLEX` 12.6.3 `Java` 12 and the construction heuristic and local searches are written in `DELMIA Quintiq` 2020 refresh 2. The `CPLEX` model was run on an Intel i7 7700HQ and the `DELMIA Quintiq` solutions were produced by an i7 5600U for 30 vessels and by a Xeon E5-2690 for 75 vessels. Various tests indicated that Xeon and the 5600U performed similarly, so we neglect the possibility of a performance gap between the two computers. The exact model is limited to an hour of solution time. All local search heuristics of 30 vessels are run for 60 seconds and for an hour for 75 vessels. The construction heuristic is able to produce a solution in a fraction of a second.

Several computational experiments are conducted to obtain the optimal local search settings, which can be found in Section A.2. Here, each parameter is chosen which scores the lowest objective over all 30 instances of size 30. For SWO, a diversification limit N_{div} of ten tries is found to be optimal. For the GA, a population size of 30 is chosen and a mutation probability for each child of $p_{\text{mutate}} = 0.8$. For the ACO the population size is set to 20 to obtain more iterations, which is necessary to converge. Furthermore, an α of 2.1 and a ρ of 0.4 was found to yield the best results.

6.2 Valid inequalities

To evaluate the influence of the proposed valid inequalities on the exact model, we run the model for 20 instances of 15 vessels. First we use the model without valid inequalities, then with inequalities (32), (33) and (34) separately and then with all the inequalities. Instances of 15 vessels are chosen as this size is challenging enough while allowing for some optimal solutions to be found. We limit the solution time of the model to one hour and indicate zero cost solutions with a dash. Only zero cost solutions were found within the time limit.

The results listed in Table 3 show that all valid inequalities decrease the average upper bound. For valid inequalities (32) and (33) this was not significant with a t-test indicating different means with a p-value of 0.15 and 0.24. For valid inequalities (34) and (32)+(33)+(34) the reduction was significant with a p-value of 0.01 and 0.04 respectively. Although all three inequalities seem to have (some) effect on the obtained upper bound, the bundling of them therefore does not seem to help considerably. Nevertheless, adding all three inequalities does improve the average upper bound by around 13%. The effect seems to be mixed as for four instances the upper bound increases over the original formulation, as indicated in bold. For other instances such as `i2n15` and `i16n15` the inequalities seems to work much better, which highlights the arbitrariness of using an exact solver. Nevertheless, the formulation with all inequalities added is able to obtain the optimal zero cost solutions much quicker. Where the original formulation obtained these solutions in 1200 seconds on average, all inequalities helped to reduce this number to 680 seconds on average. Therefore, the use of these inequalities can bring serious practical benefits for quickly obtaining low cost solutions.

Table 3: Results of the valid inequalities added to the model.

Instance	Objective	**(32)** Objective	**(33)** Objective	**(34)** Objective	**(32)+(33)+(34)** Objective	Maximum share of valid inequalities
i1n15	10,065	19,817	19,011	9,682	**19,585**	5%
i2n15	24,699	16,144	23,684	7,585	8,671	0%
i3n15	34,483	24,792	32,817	21,533	31,935	1%
i4n15	-	-	1,932	-	-	0%
i5n15	9,346	7,651	9,920	7,149	**9,846**	3%
i6n15	8,460	10,218	4,304	8,154	**14,765**	1%
i7n15	27,828	33,871	34,179	32,894	22,584	1%
i8n15	2,021	7,495	2,518	2,598	1,487	2%
i9n15	-	-	-	523	-	0%
i10n15	18,395	22,833	21,864	22,378	15,389	2%
i11n15	-	-	-	-	-	0%
i12n15	31,858	19,238	23,014	23,150	26,458	0%
i13n15	51,077	43,266	59,704	40,076	48,069	2%
i14n15	-	-	-	-	-	1%
i15n15	23,918	32,897	22,878	20,188	18,007	3%
i16n15	37,597	27,491	16,992	25,103	21,270	1%
i17n10	20,953	18,488	18,268	17,833	15,181	1%
i18n15	69,829	39,183	52,124	50,107	64,440	1%
i19n15	3,329	4,437	5,241	4,179	2,449	0%
i20n15	8,031	10,756	9,675	10,345	**11,012**	1%
Average	19,094	16,929	17,906	15,174	16,557	

To validate that not too many rows are added to the problem, we included the share all three valid inequalities take up in the model. For all instances, these remained limited to only a few percent of the total number of rows. Therefore, any potential negative side-effects of newly added rows on the performance might not be a problem. As we suspect that bundling the inequalities results in a stronger formulation and the difference between adding (34) and (32)+(33)+(34) is not significant, we use the model with all three valid inequalities in the next section.

6.3 Heuristic performance validation

We now turn to the performance of the heuristics. As these heuristics are not guaranteed to find the optimal solution due to the way the construction heuristic works, we want to validate its performance. We therefore now continue to compare the quality of the heuristic solutions to solutions of the exact model including valid inequalities. The heuristics are used without stacker-reclaimer scheduling as the exact model does not include this element. Instances of ten vessels are used as this size allows some instances to be solved to optimality while remaining challenging enough for the heuristics.

The results of the exact model with valid inequalities and the three local searches are listed in Table 4. The solution time of the optimal instances solved within an hour are indicated in bold. Moreover, we list the difference between the average of the SWO, GA and ACO solutions and the best integer solution of the model. What can be seen is that for most instances with an optimal solution, the heuristics find a competitive solution often equal or very close to the optimal solution. For harder instances such as `i8n15`, `i11n15` and `i12n15` the difference was kept limited by the heuristics by around 1000 to 2000. For the other instances, the average difference can be somewhat larger with a maximum of 5450 for instance `i15n15`. We neglect the comparison for instances with a negative difference, as the exact model was not able to find a good solution within the time limit. Nonetheless, we can conclude that using the exact model in these instances is less competitive than using the heuristics. In the end, we see an average 27%, 20% and 14% objective increase over the exact model for the SWO, GA and ACO respectively. Whether this performance gap translates into larger instances is hard to say, but for these instance sizes, the gap between the model and heuristics seems to stay limited.

Lastly, we observed that the reason for the performance gap was mostly the cause of a bad

Table 4: Results of the squeaky wheel optimisation (SWO), genetic algorithm (GA) and ant colony optimisation (ACO) compared to the exact model solutions. The percentages in the average row are the increase over the exact model objective.

	CPLEX Model		SWO	GA	ACO	
Instance	Objective	Time (seconds)	Objective	Objective	Objective	Average difference
i1n10	-	**7**	-	-	295	**98**
i2n10	11,504	3,600	16,876	16,591	13,373	4,109
i3n10	27	3,600	2,390	4,231	4,231	3,591
i4n10	12,732	3,600	10,573	10,573	16,306	(248)
i5n10	-	**11**	343	343	753	**479**
i6n10	461	3,600	1,668	832	2,060	1,059
i7n10	-	**31**	-	-	-	**0**
i8n10	138	**742**	2,565	1,560	1,560	**1,757**
i9n10	**-**	**125**	473	473	1	**316**
i10n10	12,757	3,600	12,479	14,431	15,281	1,306
i11n10	**-**	**499**	2,264	2,264	2,264	**2,264**
i12n10	922	**857**	3,367	1,618	1,618	**1,279**
i13n10	**-**	**42**	31	31	31	**31**
i14n10	**-**	**13**	**-**	**-**	**-**	**0**
i15n10	5,044	3,600	11,201	11,201	9,080	5,450
i16n10	-	**38**	-	-	89	**30**
i17n10	-	**20**	-	-	-	**0**
i18n10	5,856	3,600	11,760	11,130	10,109	5,143
i19n10	33,762	3,600	27,872	23,419	16,015	(11,327)
i20n10	1,572	3,600	4,063	3,362	3,362	2,024
Average	4,239	1,741	5,396 (+27%)	5,103 (+20%)	4,821 (+14%)	868

stockyard fit. In the heuristics, a bad stockpile placement often caused an arriving vessel to wait, incurring demurrage costs. Further analysis Appendix A.3 confirms these observations. This analysis points out that the reason for a large average difference in some instances could be the number of stockpiles carried per vessel. The more a vessel brings in, the more the heuristics seems to struggle relatively compared to exact model solutions. This indicates that instances with more stockpiles per vessel require a better fitting strategy within the stockyard. Clearly, exact solvers benefit from an integrated approach to fit in the stockpiles while these sequential search heuristics are limited in the way the construction heuristic develops the stockyard fit and the schedule.

6.4 Local search heuristics

In this section, we leave the exact model and discuss the main results of SWO, GA and ACO for instances of 30 vessels and 75 vessels. The local searches are compared to the principle of first come first serve (FCFS). This means the vessels are inserted in the order of their ETA, which is typical for a real-life planner to do. First, the average computation time and the number of iterations are listed. Then the total, vessel cancellation, demurrage and speedup costs are listed. Furthermore, more detail is given about the number of cancelled vessels, the number of delayed vessels, the total delay over all vessels and the average delay per vessel.

6.4.1 Local search - 30 vessels

The results for instances of 30 vessels are listed in Table 5. SWO, GA and ACO are all able to significantly improve the FCFS solutions, all achieving around a 30 percent cost saving. SWO obtains the lowest average cost and with that the best improvement over FCFS. A one-sided t-test for differing means shows that SWO performs significantly better than the ACO. For the difference between SWO and GA, this hypothesis is not proven at a five percent level at a p-value of 0.10. Moreover, SWO performs best for half of the instances, while GA and ACO perform best each in a quarter of the instances. As can be seen in Table 5, the total costs of all solutions are mainly demurrage and vessel cancellation. Speedup of vessel is rarely used as this

Table 5: Average results of squeaky wheel optimisation (SWO), genetic algorithm (GA) and ant colony optimisation (ACO) for instances of 30 vessels.

	FCFS	Squeaky wheel optimisation	Genetic algorithm	Ant colony optimisation
Time (seconds)	0.32	60	60	60
Iterations	1	23.7	8	7.9
Cost				
Total	69,093	46,980	47,993	48,990
Cancellation penalty	25,332	16,194	14,246	20,426
Demurrage	43,707	30,786	33,747	28,537
Speedup	53	0	0	27
Improvement over FCFS		-32%	-31%	-29%
Shipping delays				
Vessels canceled	2.0	1.1	1.0	1.4
Vessel delayed	18.1	17.0	18.0	15.9
Total delay (hours)	1,399	1,202	1,151	1,110
Average delay (hours)	46.6	40.1	38.0	37.0

is only better to do when its stockpile can fit just before another stockpile in time, so to the left in a space-time diagram. More specifically, we notice that in all instances delayed vessels have to wait with unloading, because first space needs to be freed up in the stockyard. These vessels unload their stockpiles later at a berth, which causes other vessels having to wait for these vessels, creating a chain of delayed vessels and making a speed-up action useless.

More specifically, the local search objectives are smaller than the FCFS objectives due to the number of cancelled and delayed vessels decreasing. Most significantly, the number of cancelled vessels decrease for all local searches from 2 vessels with FCFS to an average of 1.1 for SWO. Cancellations contribute more than a third to the objective as can be seen in the table, so the significant drop in cancellations contribute around 13% points to the average SWO cost saving over FCFS. The other 19% points are a result of a drop in total vessel delay. This reduction can be described by a slight decrease in total vessel delay. However, the reduction is also a result of a more limited vessel delay for large vessels with high demurrage cost. To illustrate, in FCFS the average hour delay costed $\frac{43{,}707}{1399} \approx 31$ while in SWO it costed $\frac{30{,}786}{1202} \approx 26$, which is a result of reducing the delay of larger, more expensive vessels. All three local searches obtain very similar service levels, although ACO achieves significantly less total delay than SWO with a p-value of 0.03. When one would prioritise delay over cancelled vessels, we would thus recommend ACO for these instances.

Lastly, we see that SWO obtains far more iterations than the GA and ACO. This is a consequence of the workings of the algorithms. We should thus only compare the iterations of GA with ACO. We also note that the low iteration count for the GA and ACO may seem low, although the time limit still allowed for both local searches to converge. The low iteration count is therefore not a problem for the workings of these local searches.

6.4.2 Local search - 75 vessels

For instances of 75 vessels, we see in Table 6 that in this case, all follow-up methods are able to improve the solutions of the FCFS significantly as well. With a lower average improvement though, the local searches perform a little bit less well. This could be a a result of the fact that the parameters are tweaked to instances of 30 vessels, but it could also be that the sizes of these instances are harder to improve. The ACO is the method that takes the biggest hit in performance for these larger instances. We suspect that this is the result of the value of α, as ACO often turns into a random search when this parameter is not set correctly. More specifically, we note that vessel orders with high costs take longer to evaluate for the construction heuristic. This is a consequence of cancelled vessels taking longer to be inserted or cancelled than other vessels. Therefore, difficult vessels orders containing cancellations take longer for the algorithm,

Table 6: Average results of squeaky wheel optimisation (SWO), genetic algorithm (GA) and ant colony optimisation (ACO) for instances of 75 vessels.

	FCFS	Squeaky wheel optimisation	Genetic algorithm	Ant colony optimisation
Time (seconds)	3.59	3600	3600	3600
Iterations	1.0	52.6	47.5	36.6
Cost				
Total	184,932	137,409	133,526	153,660
Cancelation penalty	72,946	47,752	45,119	66,302
Demurrage	111,947	89,632	88,335	87,302
Speedup	40	24	72	56
Improvement over FCFS		-26%	-28%	-17%
Shipping delays				
Vessels canceled	5.5	3.4	3.2	4.8
Vessel delayed	47.8	49.5	50.0	46.4
Total delay (hours)	3,558	3,446	3,247	3,367
Average delay (hours)	47.4	46.0	43.3	44.9

which is the reason why ACO obtains considerably less iterations than GA. Moreover, for these instances the GA is the best performing method, which could be a consequence of the larger instances. It could be that the GA benefits from a more general search instead of the SWO that has to go through a longer list of vessels in the intensification step. Finally, as the GA performs best in total delay and average number of cancelled vessels, this method works the best for these instances.

6.5 Instance evaluation

In Section 6.3 it became clear that in some instances the heuristics yield significantly higher objectives and may thus be harder to solve. To evaluate what problem properties challenge the heuristics, we now proceed with an analysis starting with three main problem properties that might increase the complexity of the problem. First, we recognize the total average stockpile load on each vessel as a potential candidate for making an instance more challenging. Inserting a vessel that has more load on board might be more challenging because all its content need to fit within the stockyard to not cancel the vessel. Secondly, we see instances with stockpiles having a long request time as a contender, as these piles take more usable space within the space-time diagram and thus cause a bottleneck. Thirdly, it might be that vessels arriving closely next each other might cause problems not only at the berths, but also within the stockyard, because their stockpiles need to be in the stockyard at the same time. Therefore, we also include a measure of vessel ETA proximity in our analysis.

To test whether these three instance properties make the problem harder, we plot the average value of each property against the average objective obtained by SWO, GA and ACO for each instance. The average of all local searches are taken to flatten out any incidental positive or negative results of a local search. Moreover, we add a trend line to the plot using a least square regression to see if the relation is significant.

As can be seen in Figure 13, for all three properties, the trend line is sloping up and down as expected: higher objectives when a vessel contains more load and when stockpiles need to dwell in the pad for longer. Likewise, a lower objective when the vessels arrive relatively far away from each other. As can be seen, the significance of the relation is not visible for stockpile request time and vessel proximity. Further tests in Appendix A.4 are performed with more varying instances to search for any possible significance. These indicate further insignificance, which is not the result of limited variability in both the stockpile request time and vessel proximity. The only visible relation is the stockpile load with an $R^2 = 0.65$ and a p-value for the slope estimator of 7.53E-08. To further investigate what causes a higher objective we proceed with making a further distinction within the total vessel load. A total vessel load can be divided into

an average stockpile volume and an average number of stockpiles per vessel. Hence, we proceed with the same analysis for these properties.

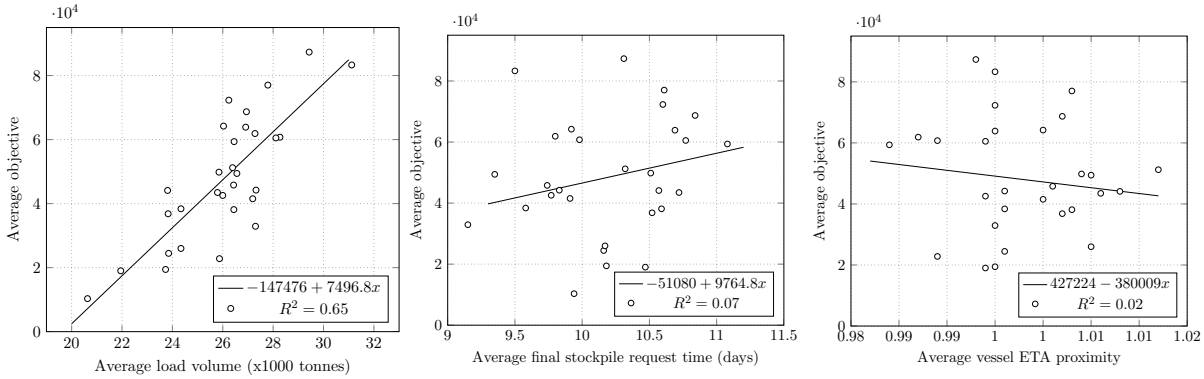


Figure 13: Average of the SWO, GA and ACO objectives versus the average stockpile load per vessel, final stockpile request time and vessel ETA proximity of each instance.

In Figure 14 we plot the average stockpile volume and the average number of stockpiles per vessel against the average objective value. For both relations we see a positive and significant trend with a p-value of 0.03 and 7.63E-09 for the average stockpile volume and average number of stockpiles per vessel respectively. We note that the average stockpile volume has a less clear trend than the average number of stockpiles per vessel, although the cause of this lays within the problem data: the stockpile volume is much more concentrated around the average volume within the problem than the number of stockpiles that need to be inserted. For instance, the maximum stockpile volume is only 6% away from the average value, while the maximum number of stockpiles per vessel is 14% away for these instances. Therefore, the effect of the number of stockpiles per vessel is much more pronounced within the instances.

Nevertheless, we can conclude that main drivers for higher heuristic objectives within the instances seem to be the average stockpile volume and the number of stockpiles per vessels. This does not come to surprise as we already concluded in Section 6.4 that a main cost driver is delayed vessels due to a stockyard bottleneck. Clearly, more and larger stockpiles that need to be fitted within the stockyard seem to complicate the problem for the heuristics. The question whether this is due to a bad placement strategy or due to the stockyard being full without any useless spaces was partly answered in Section 6.3 for smaller instances. Indeed, for instances with more stockpiles, the exact model was able to produce better results, thus hinting at a inferior placement strategy of the heuristics instead of the stockyard being full.

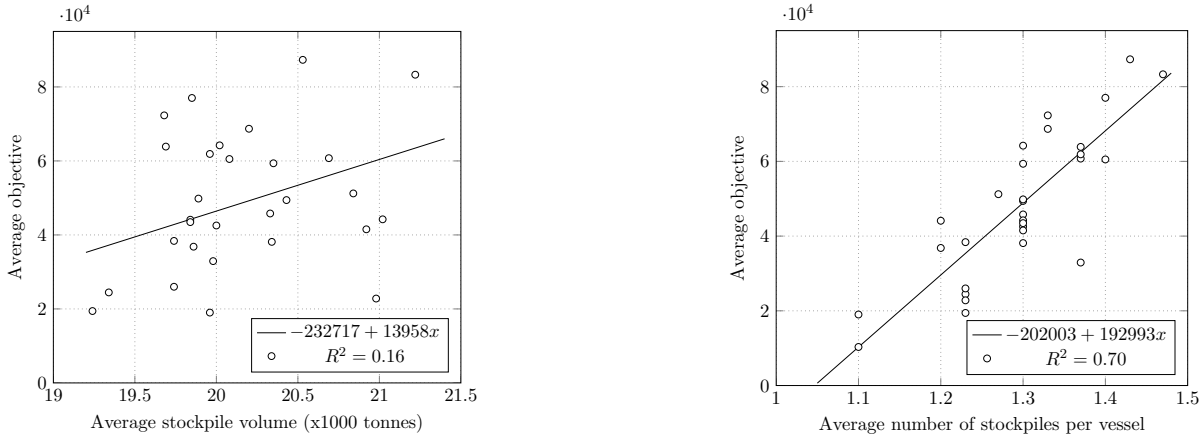


Figure 14: Average of the SWO, GA and ACO objectives versus the average stockpile volume and average number of stockpiles per vessel of each instance.

6.6 Best practices

Now we know what complicates instances for the heuristics, we are finally interested to see if any of the three methods is able to cope with these properties in the best way. To know which method produces relatively good results, we subtract the average objective of the SWO, GA and ACO from a method's found objective for each instance. In this way, negative differences indicate a better result for the local search method and a positive difference a relatively worse result. Again, we plot each instance along with a trend line.

The results for the number of stockpiles can can be seen in Figure 15. Although the fitted lines might indicate a relation, none of the relations are deemed to be significant. With a low R^2 and a p-value of 0.12, 0.47 and 0.56 for SWO, GA and ACO respectively, we conlcude that no local search is significantly better equipped in scenarios with more or less stockpiles per vessel. For the instance property of average stockpile volume, the relations were also found to be insignificant.

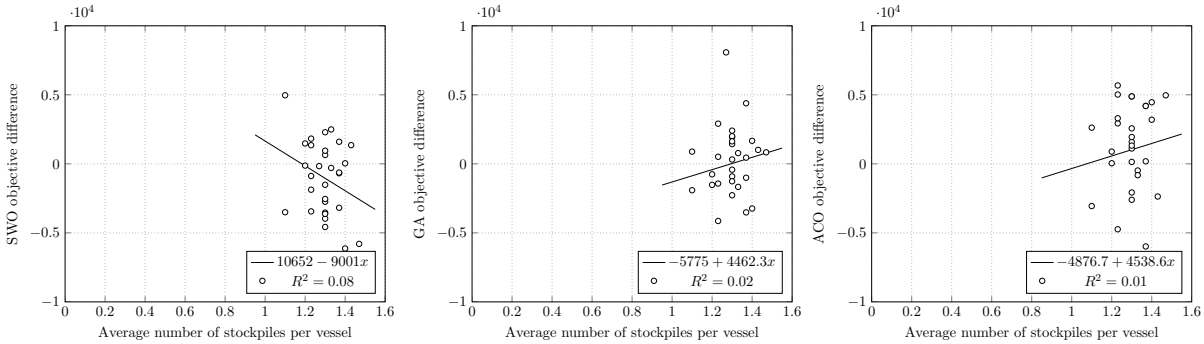


Figure 15: Difference between each method's objective and average of the SWO, GA and ACO objectives versus the average number of stockpiles per vessel of each instance.

Although all three local searches possess unique characteristics, these are therefore not enough to make a difference in performance in differing scenarios. To find a best practice for different scenarios one might thus have to step away from searching in vessel order and look at other ways in which we optimise the problem. For heuristics methods, these may include searching for a good stockpile placement strategy by searching for the best priority rules as discussed in Section 4.3.2. These priority rules could be stockpile specific, which opens the possibility of looking for the best placement for each given stockpile. For now, we confirm the result of Section 6.4.1 that the local searches are similar in performance for different instances.

6.7 Stockpile split analysis

In Section 3.4 we discussed the possibility of splitting stockpiles into two. The idea behind this is that by splitting piles that would otherwise not fit in the stockyard and placing them at two different locations, we are better able to use the stockyard available capacity. Two cautionary notes were made on splitting stockpiles in that section. First, the resulting two stockpiles each need to be separated by buffer space l_{buffer}. Secondly, as a result of the shape of the stockpile, splitting a pile results in collectively more used space in the stockyard.

To analyse whether it is advantageous to split stockpiles, we use the construction heuristic based on ETA order on all instances of 30 vessels. With this method, we gather the average objective, computation time and average number of split stockpiles over the instances. Additionally, we need to decide when to split a stockpile into two. As the stockpile insertion is always able to find a place in the space-time diagram by moving a stockpile allocation later in time, we need to decide in which conditions a split will be attempted. To limit vessel demurrage, we choose to split a stockpile based on when it would depart without splitting its piles. For this,

we first insert the vessel without splitting, which gives us an end time e_v. With this insertion, we measure the number of days n_v after a vessel's LFT_v at which the vessel $v \in V$ would depart without a split, $n_v = e_v - \text{LFT}_v$. If a vessel departs n_v days after its LFT without splitting, n_v should not be greater than a chosen amount of days $n^{\max}$. When $n_v \geq n^{\max}$, we split its stockpiles to limit its demurrage.

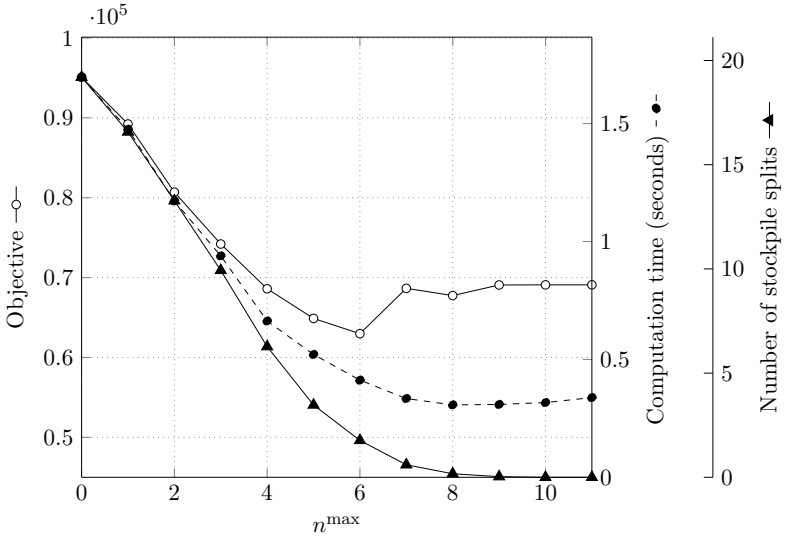


Figure 16: Effect of the maximum number of days after the LFT after which a vessel is allowed to depart without a stockpile split on the average instance objective, computation time (seconds) and number of split stockpiles

The effect of varying $n^{\max}$ from zero to eleven days is given in Figure 16. When splitting a stockpile when the vessel departs at or after its LFT at $n^{\max} = 0$, the average objective is much higher than the standard 69,093 baseline achieved by FCFS without splitting. The cause for this cost are the two reasons mentioned before, but also the stacker-reclaimers becomes a limiting factor as now two stockpiles need to be reclaimed and stacked at some point. When limiting possibilities of splitting to more days after the LFT, we see that the average objective drops down significantly below the baseline of FCFS. From the figure, it is most beneficial to split stockpiles when vessels would depart more than six days after its LFT without splitting. With this policy, on average two stockpiles are split. We also notice under this policy, that in some instances it appears that a small number of split stockpiles are placed next to each other. Although this uses costly stacker-reclaimer time, it also brings the benefit that both stockpiles are stacked and reclaimed twice as fast, which helps stockpiles to be fit more tightly.

When we limit the possibility of splitting even more, we see that the average objective increases and converges to the baseline cost of FCFS at exactly 69,093. The reason for the objective to converge to the average objective of FCFS is that with a more restricted splitting policy we see that a decreasing number of stockpiles are split. At ten and eleven days after the LFT, the number of stockpiles that are split is zero and the procedure is equivalent to the FCFS approach without splitting. Lastly, we notice that the average computation time of the construction heuristic also decreases with a more restricted split policy. This is due to the fact that less stockpiles have to be split and inserted into the stockyard for a second time, which uses computation time in the form of the insert stockpile procedure.

Further analysis in Appendix A.5 indicate that the optimal policy of six days is beneficial to implement also within the local searches. Interestingly, SWO obtains a lower average objective than without splitting, but the GA and ACO do not. We suspect that is due to the fact that the average computation time increases from 0.31 to 0.62 seconds for each evaluation of a vessel order. This prohibits a large number of evaluation of large populations for these methods.

7 Conclusion

In this thesis we considered the stockyard planning problem, which consists of several well-known problems. In the SPP defined in this study, the goal was to create a berth allocation, stockpile assignment and stacker-reclaimer schedule which minimises the cancellation, demurrage and speed-up costs over all vessels. In this schedule, the stockpiles were not allowed to overlap or mix and we had to respect the maximum number of stacker-reclaimers that were available at any time. High quality heuristics for the SPP is of great value for stockyard planners as it not only contains several problems, but also because the need for efficient use of limited quay and stockyard space is rising due to increasing dry bulk import and environmental restrictions.

Due to its increasing demand, some research has been performed on tackling the stockyard planning problem. Nevertheless, we identified four shortcomings in the literature which we try to address. First, we added the concept of partial reclaiming to stockpiles, which considerably increases the applicability of the problem. Secondly, we noticed a clear distinction between work focused on exact solutions and heuristic approaches. While we concluded that the SPP creates quite a challenge to solve without any mathematical programming techniques, we found that using an exact formulation can be beneficial for two reasons. Not only did the the exact model help to evaluate the performance of heuristic approaches, it also gave us insight into the strengths and weaknesses of the presented heuristics. Secondly, we saw the current efforts to improve construction heuristic solutions as marginal, so several follow-up methods were tested to compare their competitiveness within the same context with each other. Lastly, we tried to analyse which instance properties make the problem harder to better verify the challenge in the SPP and to find strength and weaknesses in our presented follow-up methods.

To use the exact formulation, first valid inequalities were introduced and tested to see whether they had an impact on the best found upper bound of the problem. All three inequalities together were able to decrease average integer objective significantly, which led us to adding the valid inequalities for the heuristics evaluation. When comparing the results of the heuristics with the solutions of the exact model, we saw that for instances with an optimal values, the heuristics performed remarkably well. For harder instances, we concluded that there is a performance gap for some instances between the heuristics and the MILP. Further research in the performance gap indicated that the number of stockpiles in each instance could be the main cause for the heuristic to perform relatively worse. This indicated that the resulting stockpile placement strategy of the heuristics can be further improved.

Furthermore, we tested the performance of the follow-up methods between each other on larger instances of 30 and 75 vessels. All methods achieved a significant objective reduction with the SWO being the best for 30 vessels. To then test why some instances resulted in a higher objective than other instances, we performed an analysis on a handful of instance properties. These indicated that the average stockpile size and the number of stockpiles were related with the average objective, pointing at a possible causal relationship. We argued that therefore, the heuristics could benefit from an improved stockpile placement strategy as also confirmed by the results from the exact model.

With the trouble-making instance property in hand, we finally tried to see if some methods were better able to cope with a large number of stockpiles to be inserted. In the end, no local search was able to benefit from more or less stockpiles in an instance. We argue that therefore, one should look at other optimisation approaches such that the improvements are achieved in a different way. For example, focusing on improving stockpile priority rules and fitting strategies. Lastly, we analysed whether and when it is beneficial to split stockpiles into two. We concluded that there indeed is a policy for which the construction heuristic obtains a lower average objective value. Moreover, with this policy a better average objective with SWO than without splitting was found.

We now discuss the key recommendations for stockyard planners and further research. First, for any future heuristic around stockyard planning, we stress the importance of using an exact formulation for comparison. This study was be able to quantify the performance gap between the model and heuristic approaches. And as the SPP is a complex problem due to the entanglement of vessel delay and stockpile placement strategy, it is essential to validate the outcome of heuristic methods. Secondly, to improve traditional planning solutions based on first come first serve, we prescribe the use of a follow-up method in terms of a local search in vessel order. As all local searches were able to significantly reduce the overall objective, it is worthwhile to invest time in such search procedures. As results varied depending on the instance size, we also suggest to invest sufficient time in the tweaking of the parameters depending on the instances. Thirdly, as our analysis indicated that the heuristic are challenged by scenarios with more and larger stockpiles, we suggest to also investigate what can be done to further improve the stockpile placement strategy in these instances. Lastly, for ports in which stockpile splitting is routine, we strongly suggest to investigate at what policy a split has to be performed, to avoid a scattered stockyard field. Moreover, depending on the heuristic implementation, computation times of the construction heuristic may vary. Therefore, one should be careful that an increased computation time should not result in a decrease in objective within the follow-up methods.

In the mean time, we also address three shortcomings in this study. First, we note that this problem is mainly focused on the costs of the incoming vessels. This SPP can thus mainly be characterised as an inbound stockyard planning problem. As the stockyard connects two processes at both ends, future research should thus also consider the outbound costs from the stockyard. This not only increases the potential of the developed methodology, but it also equalises the different SPPs within the literature. In all research either the SPP is inbound or outbound focused, which leads to a division within the problem. Secondly, regarding the results, the performance validation using small instances with the exact formulation is only an indication of how the performance gap will be in larger instances. Therefore, further development is needed in either exact formulations or in an alternative approach such as column generation, using for instance a set covering problem. This enables to obtain competitive lower bounds for comparison with heuristic approaches. Lastly, we noticed that the stockpile placement strategy is the biggest challenge for the heuristics. Therefore, a promising direction seems to be to further improve the stockpile placement strategy by optimising the priorities of the stockpile contact points within the procedure. More specifically, the problem can be optimised by searching for the best stockpile placement for each stockpile individually, jointly increasing the stockyard fit and decreasing the total vessel demurrage.

Acknowledgements

I would like to thank Ab Ovo for giving me the opportunity to write my thesis even when the future was unknown. In particular, I would like to thank Martin van Meerkerk for the freedom and flexibility he weekly offered me during the writing process. Without his feedback and sharp eye for detail I could not have completed the thesis in its final form. Also, I thank everyone at Ab Ovo for welcoming me and helping with answering any (unrelated) questions. In addition, I want to thank dr. D. Galindo Pecin for his supervision and charging me with new ideas to further improve my work.

References

Babich, A. and Senk, D. (2013). Coal use in iron and steel metallurgy. In *The coal handbook: towards cleaner production*, pages 267–311. Elsevier.

Babu, S. A. I., Pratap, S., Lahoti, G., Fernandes, K. J., Tiwari, M. K., Mount, M., and Xiong, Y. (2015). Minimizing delay of ships in bulk terminals by simultaneous ship scheduling, stockyard planning and train scheduling. *Maritime Economics & Logistics*, 17(4):464–492.

Belov, G., Boland, N., Savelsbergh, M. W., and Stuckey, P. J. (2014). Local search for a cargo assembly planning problem. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 159–175. Springer.

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

Bierwirth, C. and Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 244(3):675–689.

Boland, N., Gulczynski, D., and Savelsbergh, M. (2012). A stockyard planning problem. *EURO Journal on Transportation and Logistics*, 1(3):197–236.

Chen, P., Fu, Z., and Lim, A. (2002). The yard allocation problem. In *AAAI/IAAI*, pages 3–8.

Cordeau, J.-F., Laporte, G., Legato, P., and Moccia, L. (2005). Models and tabu search heuristics for the berth-allocation problem. *Transportation science*, 39(4):526–538.

Correcher, J. F., Alvarez-Valdes, R., and Tamarit, J. M. (2019). New exact methods for the time-invariant berth allocation and quay crane assignment problem. *European Journal of Operational Research*, 275(1):80–92.

Eurostat (2020). Coal production and consumption statistics. `https://ec.europa.eu/eurostat/statistics-explained/index.php/Coal_production_and_consumption_statistics`.

Fraunhofer ISE (2018). Studie: Stromgestehungskosten erneuerbare energien. `https://www.ise.fraunhofer.de/de/veroeffentlichungen/studien/studie-stromgestehungskosten-erneuerbare-energien.html`.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. (1998). Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10(4):427–437.

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

Kim, B.-I., Koo, J., and Park, B. S. (2009). A raw material storage yard allocation problem for a large-scale steelworks. *The International Journal of Advanced Manufacturing Technology*, 41(9-10):880–884.

Lipovetzky, N., Burt, C. N., Pearce, A. R., and Stuckey, P. J. (2014). Planning for mining operations with time and resource constraints. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Mauri, G. R., Ribeiro, G. M., Lorena, L. A. N., and Laporte, G. (2016). An adaptive large neighborhood search for the discrete and continuous berth allocation problem. *Computers & Operations Research*, 70:140–154.

Meisel, F. and Bierwirth, C. (2009). Heuristics for the integration of crane productivity in the berth allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):196–209.

Nishimura, E., Imai, A., and Papadimitriou, S. (2001). Berth allocation planning in the public berth system by genetic algorithms. *European Journal of Operational Research*, 131(2):282–292.

Park, Y.-M. and Kim, K. H. (2005). A scheduling method for berth and quay cranes. In *Container terminals and automated transport systems*, pages 159–181. Springer.

Singh, G., Sier, D., Ernst, A. T., Gavriliouk, O., Oyston, R., Giles, T., and Welgama, P. (2012). A mixed integer programming model for long term capacity expansion planning: A case study from the hunter valley coal chain. *European Journal of Operational Research*, 220(1):210–224.

Sun, D., Meng, Y., Tang, L., Liu, J., Huang, B., and Yang, J. (2020). Storage space allocation problem at inland bulk material stockyard. *Transportation Research Part E: Logistics and Transportation Review*, 134:101856.

The Economist (2020). The travails of "ex-ilva", europe's largest steel plant. `https://www.economist.com/europe/2020/02/27/the-travails-of-ex-ilva-europes-largest-steel-plant`.

Unsal, O. and Oguz, C. (2019). An exact algorithm for integrated planning of operations in dry bulk terminals. *Transportation Research Part E: Logistics and Transportation Review*, 126:103–121.

US Energy Information Agency (2011). Coal production, selected years, 1949-2011. `https://www.eia.gov/totalenergy/data/annual/pdf/sec7_7.pdf`.

US Energy Information Agency (2020a). Coal explained: use of coal. `https://www.eia.gov/energyexplained/coal/use-of-coal.php`.

US Energy Information Agency (2020b). U.s. coal production, 2014 - 2020. `https://www.eia.gov/coal/production/quarterly/pdf/t1p01p1.pdf`.

World Bank (2020). World development indicators online database. `https://datacatalog.worldbank.org/dataset/world-development-indicators`.

Xin, J., Negenborn, R. R., and Van Vianen, T. (2018). A hybrid dynamical approach for allocating materials in a dry bulk terminal. *IEEE Transactions on Automation Science and Engineering*, 15(3):1326–1336.

A Appendix

A.1 Total length increase when splitting stockpiles

When we assume the stockpile is always longitudinal ($d > 0, c > 0$) and we look at Figure 7, we can see that the increase in length $\Delta S_L(n)$ due to splitting into n piles can be written as

$$\Delta S_L(n) = (2r + c)n - (2r + d) = (w_p + c)n - (w_p + d).$$

The first part is two times the width of each half cones $2r = w_p$ plus the new length of the prism c times the number we split the stockpile n. The second part is the length of the stockpile without splitting. When we write out the expression and then substitute the expressions for c and d as we found in Section 3.4, we get

$$\begin{aligned}\Delta S_L(n) &= (nw_p + nc) - (w_p + d) = (n-1)w_p + nc - d \\ &= (n-1)w_p + n\left(\frac{\frac{V_i}{n} - \frac{\pi}{24}w_p^3\tan(\alpha)}{\frac{w_p^2}{4}\tan(\alpha)}\right) - \left(\frac{V_i - \frac{\pi}{24}w_p^3\tan(\alpha)}{\frac{w_p^2}{4}\tan(\alpha)}\right).\end{aligned}$$

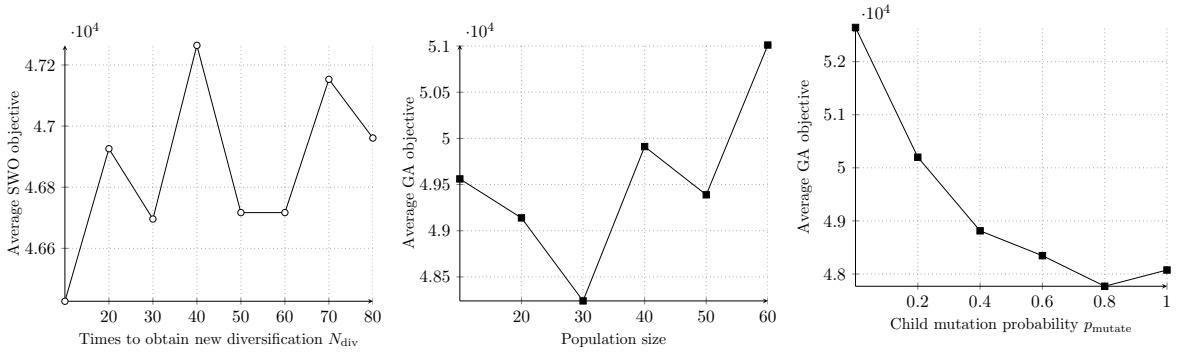
When we simplify the expression we arrive at

$$\begin{aligned}\Delta S_L(n) &= (n-1)w_p + n\left(\frac{\frac{V_i}{n} - \frac{\pi}{24}w_p^3\tan(\alpha)}{\frac{w_p^2}{4}\tan(\alpha)}\right) - \left(\frac{V_i - \frac{\pi}{24}w_p^3 w_p^3\tan(\alpha)}{\frac{w_p^2}{4}\tan(\alpha)}\right) \\ &= (n-1)w_p - \frac{(n-1)\frac{\pi}{24}w_p^3\tan(\alpha)}{\frac{w_p^2}{4}\tan(\alpha)} = (n-1)w_p - (n-1)\frac{\pi}{6}w_p \\ &= (1 - \frac{\pi}{6})w_p(n-1) \approx 0.476w_p(n-1).\end{aligned}$$

We conclude that the an extra total length of $0.476w_p(n-1)$ in the stockyard is needed to insert the stockpile when splitting the pile n times.

A.2 Sensitivity analysis

In Section 6.1 the optimal paramaters of the three local searches are noted. These parameters are found by using the local searches on all 30 instances of 30 vessels and choosing the parameter which obtains the lowest average objective. The results of this analysis can be found in Figure 17 and Figure 18.

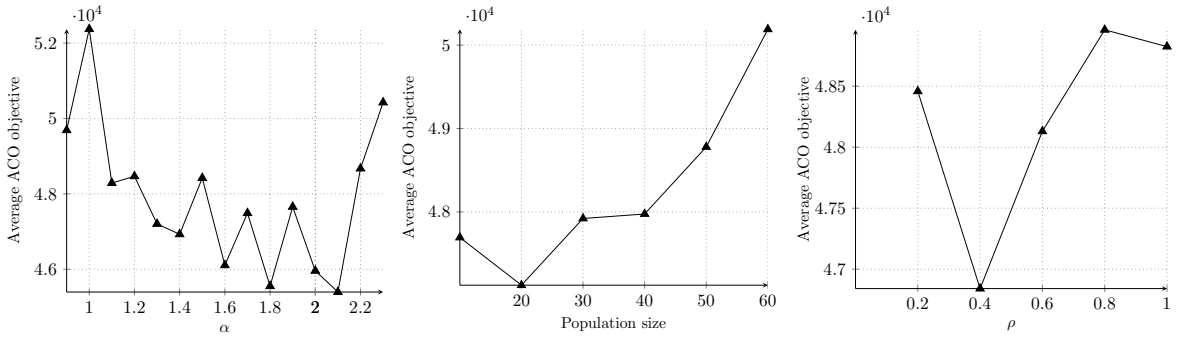


(a) Squeaky wheel optimisation.

(b) Population size GA.

(c) Child mutation probability GA.

Figure 17: Effect of different Squeaky wheel optimisation (SWO) and Genetic algorithm (GA) parameters on the average objective of all 30 instances of size 30.



(a) α in ACO. **(b)** Population size ACO. **(c)** ρ in ACO.

Figure 18: Effect of different ant colony optimisation (ACO) parameters on the average objective of all 30 instances of size 30.

A.3 Heuristics performance validation - further results

In Section 6.3 we see that for some instances relatively higher objectives are obtained for the heuristics compared the exact model solutions. To investigate what causes this gap between heuristic and model in some cases, the relation between the difference between average objective of SWO, GA and ACO and the objective of the exact model versus various instance properties as the ones described in Section 6.5 is tested. In this case, only the average number of stockpiles per vessel yields a slight significant relation as seen in Figure 19. The more stockpiles are carried by a vessel, the gap between the objectives of the heuristics and the model are on average higher. With a p-value for the slope estimator of 0.03 the relation deemed to be significant.

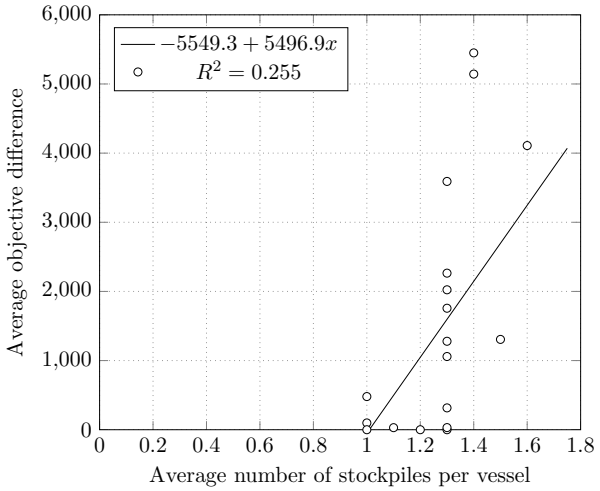


Figure 19: Difference between the average objective of SWO, GA and ACO and the objective of the exact model versus the average number of stockpiles per vessel for each instance

A.4 Instance evaluation - further results

In Section 6.5 the relations between the average heuristics objective and the average stockpile request duration and vessel ETA proximity was not significant. To test if this is due to the low variance of the independent variable, we make more extreme instances for the two properties. This is done by increasing the dwell time of the stockpiles and by increasing the variance in which ships arrive. Again, we plot the results and make a trend line in Figure 20. Although the effect of a longer stockpile request time is more significant with a p-value of 0.08, both relations are still not significant with a more diverse independent variable. Therefore, there

is no further evidence of any significant relation between the average objective and these two instance properties.

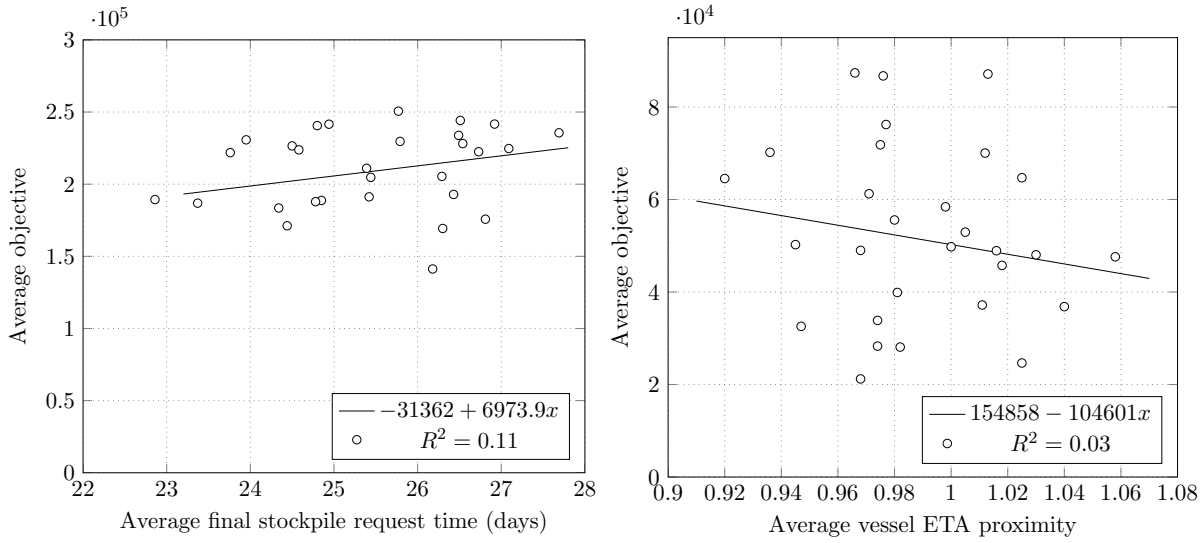


Figure 20: Average of the SWO, GA and ACO objectives versus the extreme final stockpile request time and vessel ETA proximity of each instance.

A.5 Stockpile split analysis - further results

To see if the stockpile splitting in Section 6.7 was useful in the follow-up methods, we generated results of the three local searches with splitting of stockpiles turned and $n_{\max} = 6$ for the instances of 30 vessels. Most importantly, only SWO is able to improve with stockpile splitting such that we obtain the lowest average objective of 43,722, which is lower than the SWO without splitting. For the other two methods, we do not see a significantly better average objective value than without splitting. We suspect that this is due to the fact that one construction heuristic takes more time on average with stockpile splitting turned on (0.62 vs 0.31 seconds). This limits the evaluation of large populations within the GA and ACO.

Table 7: Average results for of squeaky wheel optimisation (SWO), genetic algorithm (GA) and ant colony optimisation (ACO) for splitting stockpiles with $n_{\max} = 6$ with instances of 30 vessels.

	FCFS	**Squeaky wheel optimisation**	**Genetic algorithm**	**Ant colony optimisation**
Time (seconds)	0.62	60	60	60
Iterations	1	18.8	8.0	4.2
Cost				
Total	62,981	43,722	47,993	48,651
Cancelation penalty	22,105	13,495	14,246	17,582
Demurrage	40,860	30,227	33,747	31,069
Speedup	17	0	0	0
Improvement over FCFS		-31%	-24%	-23%
Shipping delays				
Vessels canceled	1.7	0.9	1.0	1.2
Vessel delayed	18.2	17.3	18.0	15.6
Total delay (hours)	1,307	1,127	1,151	1,093
Average delay (hours)	43.6	37.6	38.0	36.4