Local search for a cargo assembly planning problem

G. Belov[1] and N. Boland[2] and M.W.P. Savelsbergh[2] and P.J. Stuckey[1]

[1] Department of Computing and Information Systems
University of Melbourne, 3010 Australia
[2] School of Mathematical and Physical Sciences
University of Newcastle, Callaghan 2308, Australia.

Abstract. We consider a real-world cargo assembly planning problem arising in a coal supply chain. The cargoes are built on the stockyard at a port terminal from coal delivered by trains. Then the cargoes are loaded onto vessels. Only a limited number of arriving vessels is known in advance. The goal is to minimize the average delay time of the vessels over a long planning period. We model the problem in the MiniZinc constraint programming language and design a large neighbourhood search scheme. We compare against (an extended version of) a greedy heuristic for the same problem.

Keywords: packing, scheduling, resource constraint, large neighbourhood search, constraint programming, adaptive greedy, visibility horizon

1 Introduction

The Hunter Valley Coal Chain (HVCC) refers to the inland portion of the coal export supply chain in the Hunter Valley, New South Wales, Australia. Coal from different mines with different characteristics is 'mixed' in a stockpile at a terminal at the port to form a coal blend that meets the specifications of a customer. Once a vessel arrives at a berth at the terminal, the stockpiles with coal for the vessel are reclaimed and loaded onto the vessel. The vessel then transports the coal to its destination. The coordination of the logistics in the Hunter Valley is challenging as it is a complex system involving 14 producers operating 35 coal mines, 27 coal load points, 2 rail track owners, 4 above rail operators, 3 coal loading terminals with a total of 8 berths, and 9 vessel operators. Approximately 1700 vessels are loaded at the terminals in the Port of Newcastle each year. For more information on the HVCC see the overview presentation of the Hunter Valley Coal Chain Coordinator (HVCCC), the organization responsible for planning the coal logistics in the Hunter Valley [6].

We focus on the management of a stockyard at one of the coal loading terminals. It acts as a *cargo assembly terminal* where the coal blends assembled and stockpiled are based on the demands of the arriving ships. Our cargo assembly planning approach aims to minimize the delay of vessels, where the delay of a vessel is defined as the difference between the vessel's departure time and its earliest possible departure time, that is, the departure time in a system with infinite capacity. Minimizing the delay of vessels is used as a proxy for maximizing the throughput, i.e., the maximum number of tons of coal that can be handled per year, which is of crucial importance as the demand for coal is expected to grow substantially over the next few years. We investigate the value of information given by the *visibility horizon* — the number of future vessels whose arrival time and stockpile demands are known in advance.

The solving technology we apply is Constraint Programming (CP) using lazy clause generation (LCG) [11]. Constraint programming has been highly successful in tackling complex packing and scheduling problems [17, 18]. Cargo assembly is a combined scheduling and packing problem. The specific problem is first described by Savelsbergh and Smith [14]. They propose a greedy heuristic for solving the problem and investigate some options concerning various characteristics of the problem. We present a Constraint Programming model implemented in the MiniZinc language [9]. To solve the model efficiently, we develop iterative solving methods: greedy methods to obtain initial solutions and large neighbourhood search methods [13] to improve them.

2 Cargo assembly planning

The starting point for this work is the model developed in [14] for stockyard planning.

The stockyard studied has four pads, A, B, C, and D, on which cargoes are assembled. Coal arrives at the terminal by train. Upon arrival at the terminal, a train dumps its contents at one of three dump stations. The coal is then transported on a conveyor to one of the pads where it is added to a stockpile by a stacker. There are six stackers, two that serve pad A, two that serve both pads B and C, and two that serve pad D. A single stockpile is built from several train loads over several days. After a stockpile is completely built, it dwells on its pad for some time (perhaps several days) until the vessel onto which it is to be loaded is available at one of the berths. A stockpile is reclaimed using a bucket-wheel reclaimer and the coal is transferred to the berth on a conveyor. The coal is then loaded onto the vessel by a shiploader. There are four reclaimers, two that serve both pads A and B, and two that serve both pads C and D. Both stackers and reclaimers travel on rails at the side of a pad. Stackers and reclaimers that serve the same pads cannot pass each other. A scheme of the stockyard is given in Figure 1.

The cargo assembly planning process involves the following steps. An incoming vessel defines a set of cargoes (different blends of coal) to be assembled and an *estimated time of arrival (ETA)*. The cargoes are assembled in the stockyard as different stockpiles. The vessel cannot arrive at berth earlier than its ETA. Once at a berth, and once all its cargoes have been assembled, the reclaiming of the stockpiles (the loading of the vessel) begins. The stockpiles are reclaimed onto the vessel in a specified order to maintain physical balancing constraints.

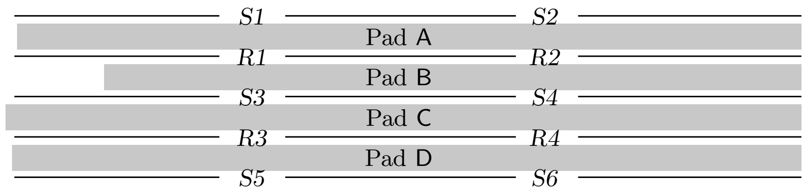


Fig. 1. A scheme of the stockyard with 4 pads, 6 stackers, and 4 reclaimers

The goal of the planning process is to maximize the throughput without causing unacceptable delays for the vessels.

When assigning each cargo of a vessel to a location in the stockyard we need to schedule the stacking and reclaiming of the stockpile taking into account limited stockyard space, stacking rates, reclaiming rates, and reclaimer movements. We model stacking and reclaiming at different levels of granularity. All reclaimer activities, e.g., the reclaimer movements along its rail track and the reclaiming of a stockpile, are modelled in time units of one minute. Stacking is modelled only at a coarse level of detail in 12 hour periods.

We assume that the time to build a stockpile is derived from the locations of the mines that contribute coal to the blend (the distance of the mines from the port). We allocate 3, 5, or 7 days to stacking of different stockpiles depending on the blend. We assume that the tonnage of the stockpile is stacked evenly over the stacking period. Since the trains that transport coal from the mines to the terminal are scheduled closer to the day of operations, this is not unreasonable. We assume that all stockpiles for a vessel are assembled on the same pad, since that leads to better results (already observed in [14]). In practice, however, there is no such restriction.

For each stockpile we need to decide a location, a stacking start time, a reclaiming start time, and which reclaimer will be used. Note that reclaiming does not have to start as soon as stacking has finished; the time between the completion of stacking and the start of reclaiming is known as *dwell time*. Stockpiles cannot overlap in time and space, reclaimers can only be used on pads they serve, and reclaimers cannot cross each other on the shared track. The waiting time between the reclaiming of two consecutive stockpiles of one vessel is limited by the *continuous reclaim time limit*. The reclaiming of a stockpile, a so-called *reclaim job*, cannot be interrupted.

A cargo assembly plan can conveniently be represented using *space-time diagrams*; one space-time diagram for each of the pads in the stockyard. A space-time diagram for a pad shows for any point in time which parts of the pad are occupied by stockpiles (and thus also which parts of the pad are not occupied by stockpiles and are available for the placement of additional stockpiles) and the locations of the reclaimers serving that pad. Every pad is rectangular; however its width is much smaller than its length and each stockpile is spread across the entire width. Thus, we model pads as one-dimensional entities. The location of a stockpile can be characterized by the position of its lowest end called its

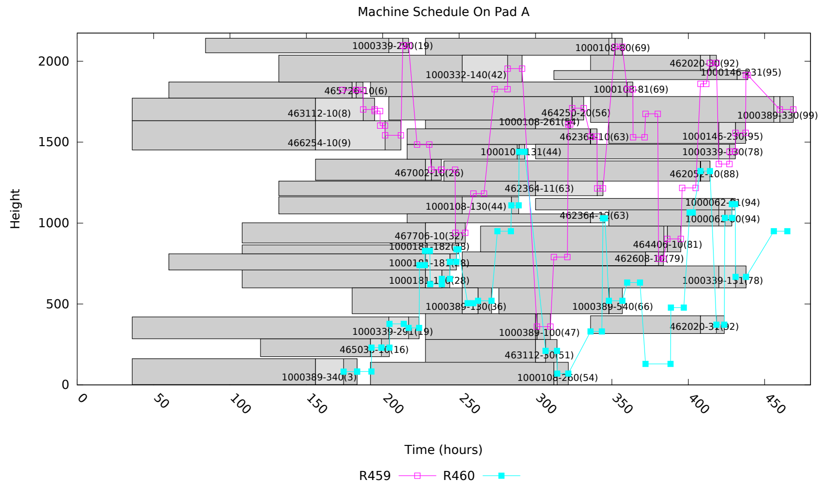


Fig. 2. A space-time diagram of pad A showing also reclaimer movements. Reclaimer R459 has to be after R460 on the pad. Both reclaimers also have jobs on pad B.

height. A stockpile occupies space on the pad for a certain amount of time. This time can be divided into three distinct parts: the *stacking part*, i.e., the time during which the stockpile is being built; the *dwell part*, i.e., the time between the end of stacking and the start of reclaiming; and a *reclaiming part*, i.e., the time during which the stockpile is reclaimed and loaded on a waiting vessel at a berth. Thus, each stockpile can be represented in a space-time diagram by a three-part rectangle as shown in Figure 2.

2.1 The basic Constraint Programming model

We present the model of the cargo assembly problem below; the structure corresponds directly to the implementation in MiniZinc [**?**]. The unit for time parameters is minutes, and for space parameters is meters. In addition, stacking start times are restricted to be multiples of 12 hours.

Parameter sets

S	—	set of stockpiles of all vessels, ordered by vessels' ETAs and reclaim sequence of each vessel's stockpiles
V	—	set of vessels, ordered by ETAs

Parameters

v_s	—	vessel for stockpile $s \in S$
eta_v	—	estimated time of arrival of vessel $v \in V$, minutes
$d^S_s \in \{4320, 7200, 10080\}$	—	stacking duration of stockpile $s \in S$, minutes
d^R_s	—	reclaiming duration of stockpile $s \in S$, minutes
l_s	—	length of stockpile $s \in S$, meters

$(H_1, \ldots, H_4) = (2142, 1905, 2174, 2156)$	—	pad lengths, meters
$\text{speed}^R = 30$	—	travel speed of a reclaimer, meters / minute
$\text{tonn}_s^{\text{daily}}$	—	daily stacking tonnage of stockpile $s \in S$, tonnes
$\text{tonn}^{\text{DIT}} = 537{,}600$	—	daily inbound throughput (total daily stacking capacity), tonnes
$\text{tonn}_k^{\text{SS}} = 288{,}000$	—	daily capacity of stacker stream $k \in \{1, 2, 3\}$, tonnes

Decisions

$p_v \in \{1, \ldots, 4\}$	—	pad on which the stockpiles of vessel $v \in V$ are assembled
$h_s \in \{0, \ldots, H_{p_{v_s}} - l_s\}$	—	position of stockpile $s \in S$ (of its 'closest to pad start' boundary) on the pad
$t_s^S \in \{0, 720, \ldots\}$	—	stacking start time of stockpile $s \in S$
$r_s \in \{1, \ldots, 4\}$	—	reclaimer used to reclaim stockpile $s \in S$
$t_s^R \in \{\text{eta}_{v_s}, \text{eta}_{v_s} + 1, \ldots\}$	—	reclaiming start time of stockpile $s \in S$

Constraints. Reclaiming of a stockpile cannot start before its vessel's ETA:

$$t_s^R \geq \text{eta}_{v_s}, \qquad \forall s \in S$$

Stacking of a stockpile starts no more than 10 days before its vessel's ETA:

$$t_s^S \geq \text{eta}_{v_s} - 14400, \qquad \forall s \in S$$

Stacking of a stockpile has to complete before reclaiming can start:

$$t_s^S + d_s^S \leq t_s^R, \qquad \forall s \in S$$

The reclaim order of the stockpiles of a vessel has to be respected:

$$t_s^R + d_s^R \leq t_{s+1}^R, \qquad \forall s \in S \text{ where } \; v_s = v_{s+1}$$

The continuous reclaim time limit of 5 hours has to be respected:

$$t_{s+1}^R - 300 \leq t_s^R + d_s^R, \qquad \forall s \in S \text{ where } \; v_s = v_{s+1}$$

A stockpile has to fit on the pad it is assigned to:

$$0 \leq h_s \leq H_{p_{v_s}} - l_s, \qquad \forall s \in S$$

Stockpiles cannot overlap in space and time:

$$p_{v_s} \neq p_{v_t} \vee h_s + l_s \leq h_t \vee h_t + l_t \leq h_s \vee t_s^R + d_s^R \leq t_t^S \vee t_t^R + d_t^R \leq t_s^S, \qquad \forall s < t \in S$$

Reclaimers can only reclaim stockpiles from the pads they serve:

$$p_{v_s} \leq 2 \Leftrightarrow r_s \leq 2, \qquad \forall s \in S$$

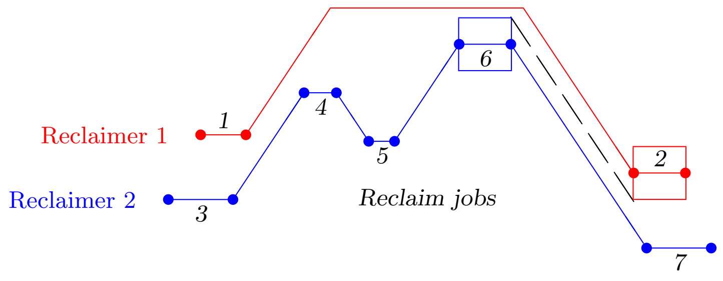


Fig. 3. A schematic example of space (vertical)-time (horizontal) location of Reclaimers 1 and 2 with some reclaim jobs. Reclaimer 2 has to stay spatially before Reclaimer 1.

If two stockpiles $s < t$ are reclaimed by the same reclaimer, then the time between the end of reclaiming the first and the start of reclaiming the second should be enough for the reclaimer to move from the middle of the first to the middle of the second:

$$r_s \neq r_t \vee \max\{(t_t^R - t_s^R - d_s^R), (t_s^R - t_t^R - d_t^R)\}\, \text{speed}^R \geq \left| h_s + \frac{l_s}{2} - h_t - \frac{l_t}{2} \right|$$

To avoid clashing, at any point in time, the position of Reclaimer 2 should be before the position of Reclaimer 1 and the position of Reclaimer 4 should be before the position of Reclaimer 3. An example of the position of Reclaimers 1 and 2 in space and time is given in Figure 3 (see also Figure 2). Because Job 3 is spatially before Job 1, there is no concern for a clash. However, since Job 6 is spatially after Job 2, we have to ensure that there is enough time for the reclaimers to get out of each other's way. The slope of the dashed line corresponds to the reclaimer's travel speed (speed^R), so we see that the time between the end of Job 6 and the start of Job 2 has to be at least $(h_6 + l_6 - h_2)/\,\text{speed}^R$.

We model clash avoidance by a disjunction: for any two stockpiles $s \neq t$, one of the following conditions must be met: either $(r_s \geq 3 \wedge r_t \leq 2)$, in which case r_s and r_t serve different pads; or $r_s < r_t$, in which case r_s does not have to be before r_t; or $h_s + l_s \leq h_t$, in which case stockpile s is before stockpile t; or, finally, enough time between the reclaim jobs exists for the reclaimers to get out of each other's way:

$$\begin{aligned}\max\{(t_t^R - t_s^R - d_s^R), (t_s^R - t_t^R - d_t^R)\}\, \text{speed}^R &\geq h_s + l_s - h_t \\ \vee\, r_s < r_t \vee (r_s \geq 3 \wedge r_t \leq 2) \vee h_s + l_s \leq h_t, &\quad \forall s \neq t \in S\end{aligned}$$

Redundant cumulatives on pad space usage improved efficiency. They require derived variables l_s^p giving the 'pad length of stockpile s on pad p':

$$l_s^p = \begin{cases} l_s, & \text{if } p_{v_s} = p, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall s \in S,\ p \in \{1, \ldots, 4\}$$

$$\texttt{cumulative}(t^S, t^R + d^R - t^S, l^p, H_p), \qquad p \in \{1, \ldots, 4\}$$

The stacking capacity is constrained day-wise. If a stockpile is stacked on day d and the stacking is not finished before the end of d, the full daily tonnage of that stockpile is accounted for using derived variables

$$t^{S1} = \lfloor t^S/1440 \rfloor, \quad d^{S1} = \lfloor d^S/1440 \rfloor$$

The daily stacking capacity cannot be exceeded:

$$\texttt{cumulative}(t^{S1}, d^{S1}, \mathrm{tonn}^{\mathrm{daily}}, \mathrm{tonn}^{\mathrm{DIT}})$$

The capacity of stacker stream k (a set of two stackers serving the same pads) is constrained similar to pad space usage:

$$\mathrm{tonn}^{\mathrm{daily}}_{ks} = \begin{cases} \mathrm{tonn}^{\mathrm{daily}}_{s}, & \text{if } (p_{v_s}, k) \in \{(1,1),(2,2),(3,2),(4,3)\} \\ 0, & \text{otherwise,} \end{cases} \qquad \forall s, k$$
$$\texttt{cumulative}(t^{S1}, d^{S1}, \mathrm{tonn}^{\mathrm{daily}}_{k}, \mathrm{tonn}^{\mathrm{SS}}_{k}), \qquad k \in \{1,2,3\}$$

The maximum number of simultaneously berthed ships is 4. We introduce derived variables for vessels' berth arrivals and use a decomposed cumulative:

$$t^{\mathrm{Berth}}_{v} = t^{R}_{s^{\mathrm{first}}(v)}, \qquad s^{\mathrm{first}}(v) = \min\{s | v_s = v\}, \qquad \forall v \in V$$
$$\mathrm{card}(\{u \in V \mid u \neq v,\ t^{\mathrm{Berth}}_{u} \leq t^{\mathrm{Berth}}_{v} \wedge t^{\mathrm{Berth}}_{v} < t^{\mathrm{Depart}}_{u}\}) \leq 3, \qquad \forall v \in V$$

Objective function. The objective is to minimize the sum of vessel delays. To define vessel delays, we introduce the derived variables t^{Depart}_{v} for vessel departure times:

$$\mathrm{depEarliest}_v = \mathrm{eta}_v + \sum_{s | v_s = v} d^R_s, \qquad \forall v \in V$$
$$t^{\mathrm{Depart}}_{v} = t^{R}_{s^{\mathrm{last}}(v)} + d^{R}_{s^{\mathrm{last}}(v)}, \qquad s^{\mathrm{last}}(v) = \max\{s | v_s = v\}, \quad \forall v \in V$$
$$\mathrm{delay}_v = t^{\mathrm{Depart}}_{v} - \mathrm{depEarliest}_v, \qquad \forall v \in V$$
$$\textbf{objective} = \sum_{v} \mathrm{delay}_v \qquad (1)$$

2.2 Solver search strategy

Many Constraint Programming models benefit from a custom search strategy for the solver. Similar to packing problems [5], we found it advantageous to separate branching decisions by groups of variables. We start with the most important variables — departure times of the ships (equivalently, delays). Then we fix reclaim starts, pads, reclaimers, stack starts, and pad positions. For most of the variables, we use the dichotomous strategy `indomain_split` for value selection, which divides the current domain of a variable in half and tries first to find a solution in the lower half. However, pads are assigned randomly, and reclaimers are assigned preferring lower numbers for odd vessels and higher numbers for even vessels. Pad positions are preferred so as to be closer to the native side of the chosen reclaimer, which corresponds to the idea of opportunity costs in [14].

In the greedy and LNS heuristics described next, some of the variables are fixed and the model optimizes only the remaining variables. For those free variables, we apply the search strategy described above.

2.3 A greedy search heuristic with Constraint Programming

It is difficult to obtain even feasible solutions for large instances in a reasonable amount of time. Moreover, even for smaller instances, if a feasible solution is found, it is usually bad. Therefore, we apply a divide-and-conquer strategy which schedules vessels by groups (e.g., solve vessels 1–5, then vessels 6–10, then vessels 11–15, etc.). For each group, we allow the solver to run for a limited amount of time, and, if feasible solutions are found, take the best of these, or, if no feasible solution is found, we reduce the number of vessels in the group and retry. We refer to this scheme as the *extending horizon* (EH) heuristic. This heuristic is generalized in Section 2.5.

2.4 Large neighbourhood search

After obtaining a feasible solution, we try to improve it by re-optimizing subsets of variables while others are fixed to their current values, a *large neighbourhood search* approach [13]. We can apply this improvement approach to both complete solutions (*global LNS*) or only for the current visibility horizon (see Section 2.5). The free variables used in the large neighbourhood search are the decision variables associated with certain stockpiles.

Neighbourhood construction methods. We consider a number of methods for choosing which stockpile groups to re-optimize (the *neighbourhoods*):

Spatial Groups of stockpiles located close to each other on one pad, measured in terms of their space-time location.

Time-based (finish) Groups of stockpiles on at most two pads with similar reclaim end times.

Time-based (ETA) Groups of stockpiles on at most two pads belonging to vessels with similar estimated arrival times.

Examples of a spatial and a time-based neighbourhood are given in Figure 4.

First, we randomly decide which of the three types of neighbourhood to use. Next, we construct all neighbourhoods of the selected type. Finally, we randomly select one neighbourhood for resolving.

Spatial neighbourhoods are constructed as follows. In order to obtain many different neighbourhoods, every stockpile seeds a neighbourhood containing only that stockpile. Then all neighbourhoods are expanded. Iteratively, for each neighbourhood, and for each direction right, up, left, and down, independently, we

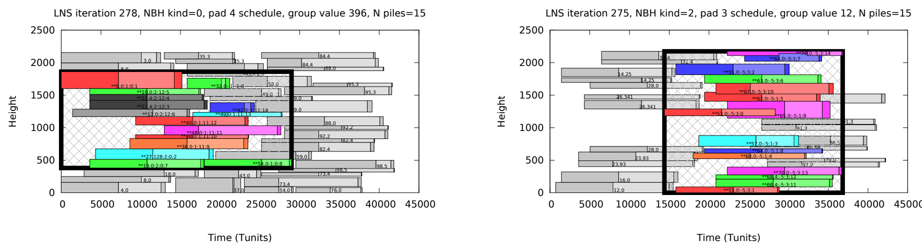


Fig. 4. Examples of LNS neighbourhoods: spatial (left) and time-based (right)

add the stockpile on the same pad that is first met by the sweep line going in that direction, after the sweep line has touched the smallest enclosing rectangle of the stockpiles currently in the neighbourhood. We then add all stockpiles contained in the new smallest enclosing rectangle. We continue as long as there are neighbourhoods containing fewer than the target number of stockpiles.

Time-based neighbourhoods are constructed as follows. Stockpiles are sorted by their reclaim end time or by the ETA of the vessels they belong to. For each pair of pads, we collect all maximal stockpile subsequences of the sorted sequence of up to a target length, with stockpiles allocated to these pads.

Having constructed all neighbourhoods of the chosen type, we randomly select one neighborhood of the set. The probability of selecting a given neighborhood is proportional to its *neighborhood value*: if the *last, but not all* stockpiles of a vessel is in the neighborhood, then add the vessel's delay; instead, if *all stockpiles* of a vessel are in the neighborhood, then add 3 times the vessel's delay.

We denote the iterative large neighbourhood search method by LNS($k_{\max}, n_{\max}, \delta$), where for at most $k_{\max}$ iterations, we re-optimize neighborhoods of up to $n_{\max}$ stockpiles chosen using the principles outlined above, requiring that the total delay decreases at least by δ minutes in each iteration. The objective is again to minimize the total delay (1).

2.5 Limited visibility horizon

In the real world, only a limited number of vessels is known in advance. We model this as follows: the current *visibility horizon* is N vessels. We obtain a schedule for the N vessels and fix the decisions for the first F vessels. Then we schedule vessels $F+1, \ldots, F+N$ (making the next F vessels visible) and so on. Let us denote this approach by VH N/F. Our default visibility horizon setting is VH 15/5, with the schedule for each visibility horizon of 15 vessels obtained using EH from Section 2.3 and then (possibly) improved by LNS(30, 15, 12), i.e., 30 LNS iterations with up to 15 stockpiles in a neighbourhood, requiring a total delay improvement of at least 12 minutes. We used only time-based neighbourhoods in this case, because for small horizons, spatial neighbourhoods on one pad are too small. (Note that the special case VH 5/5 without LNS is equivalent to the heuristic EH.)

3 An adaptive scheme for a heuristic from the literature

The truncated tree search (TTS) greedy heuristic [14] processes vessels according to a given sequence. It schedules a vessel's stockpiles taking the vessel's delay into account. It performs a partial lookahead by considering *opportunity costs* of a stockpile's placement, which are related to the remaining flexibility of a reclaimer. However, it does not explicitly take later vessels into account; thus, the visibility horizon of the heuristic is one vessel. The heuristic may perform backtracking of its choices if the continuous reclaim time limit cannot be satisfied.

The default version of TTS processes vessels in their ETA order. We propose an adaptive framework for this greedy algorithm. This framework might well be used with the Constraint Programming heuristic from Section 2.3, but the latter is slower. Below we present the adaptive framework, then highlight some modelling differences between CP and TTS.

3.1 Two-phase adaptive greedy heuristic (AG)

The TTS greedy heuristic processes vessels in a given order. We propose an adaptive scheme consisting of two phases. In the first phase, we iteratively adapt the vessel order, based on vessels' delays in the generated solutions. In the second phase, earlier generated orders are randomized. Our motivation to add the randomization phase was to compare the adaptation principle to pure randomization.

For the first phase, the idea is to prioritize vessels with large delays. We introduce vessels' "weights" which are initialized to the ETAs. In each iteration, the vessels are fed to TTS in order of non-decreasing weights. Based on the generated solution, the weights are updated to an average of previous values and ETA minus a randomized delay; etc. We tried several variants of this principle and the one that seemed best is shown in Figure 5, Phase 1. The variable "oldWFactor" is the factor of old weights when averaging them with new values, starting from iteration 1 of Phase 1.

In the second phase, we randomize the orderings obtained in Phase 1. Each iteration in Phase 1 generated a vessel order $\wr = (v_1, \ldots, v_{|V|})$. Let $\mathcal{O} = (\wr_1, \ldots, \wr_k)$ be the list of orders generated in Phase 1 in non-decreasing order of TTS solution value. We select an order with index k_0 from $\mathcal{O}$ using a truncated geometric distribution with parameter $p = p_1$, TGD(p), which has the following probabilities for indexes $\{1, \ldots, k\}$:

$$P[1] = p+(1-p)^k, \quad P[2] = p(p-1), \quad P[3] = p(p-1)^2, \ldots, \quad P[k] = p(p-1)^{k-1}$$

The rationale behind this distribution is to respect the ranking of obtained solutions. A similar order randomization principle was used, e.g., in [7]. Then we

```
Algorithm AG($k_1, k_2$)
INPUT: Instance with $V$ set of vessels; $k_1, k_2$ parameters
FUNCTION rnd($a, b$) returns a pseudo-random number uniformly distributed in $[a, b)$
Initialize weights: $\mathcal{W}_v = \text{eta}_v, \quad v \in V$
for $k = \overline{0, k_1}$                                                  [PHASE 1]
    Sort vessels by non-decreasing values of $\mathcal{W}_v$,
        giving vessels' permutation $\wr = (v_1, \ldots, v_{|V|})$
    Run TTS Greedy on $\wr$
    Add $\wr$ to the sorted list $\mathcal{O}$
    Set oldWFactor = rnd(0.125, 1)                              // "Value of history"
    Set $\mathcal{D}_v$ to be the delay of vessel $v \in V$
    Let $\mathcal{W}_v = \text{oldWFactor} \cdot \big(\mathcal{W}_v + (\text{eta}_v - \text{rnd}(0,1) \cdot \mathcal{D}_v)\big), \quad v \in V$
end for
for $k = \overline{1, k_2}$                                                  [PHASE 2]
    Select an ordering $\wr$ from $\mathcal{O}$ according to TGD(0.5)
    Create new ordering $\widetilde{\wr}$ from $\wr$,
        extracting each new vessel according to TGD(0.85)
    Run TTS Greedy with the vessel order $\widetilde{\wr}$
    Add the new ordering $\widetilde{\wr}$ to the sorted list $\mathcal{O}$
end for
```

Fig. 5. The adaptive scheme for the greedy heuristic.

modify the selected order $\wr_{k_0}$: vessels are extracted from it, again using the truncated geometric distribution with parameter $p = p_2$, and are added to the end of the new order $\widetilde{\wr}$. Then TTS is executed with $\widetilde{\wr}$ and $\widetilde{\wr}$ is inserted into $\mathcal{O}$ in the position corresponding to its objective value. We denote the algorithm by AG(k_1, k_2), where k_1, k_2 are the number of iterations in Phases 1 and 2, respectively. Note that AG($k_1, 0$) is a pure Phase 1 method, while AG($0, k_2$) is a pure randomization method starting from the ETA order.

3.2 Differences between the approaches

The model used by both methods is essentially identical, but there are small technical differences: the CP model uses discrete time and space and tonnages (minutes, meters, and tons), and discretizes possible stacking start times to be 12 hours apart. The discretized stacking start times reduce the search space, and may diminish solution quality, but seem reasonable given the coarse granularity of the stacking constraints imposed. The greedy method does not implement the berth constraints. If we remove them from the CP model it is solved slower, but the delay is hardly affected, so we always include them.

4 Experiments

After describing the experimental set-up, we illustrate the test data. We present numerical results starting with the value of information represented by the visibility horizon. Using the model from Section 2.1 we compare the Constraint Programming approach to the TTS heuristic and the adaptive scheme from Section 3.

The Constraint Programming models in the MiniZinc language were created by a master program written in C++, which was compiled in GNU C++.

The adaptive framework for the TTS heuristic and the TTS heuristic itself were implemented in C++ too. The MiniZinc models were processed by the finite-domain solver Opturion CPX 1.0.2 [12] which worked single-threaded on an Intel® Core™ i7-2600 CPU @ 3.40GHz under Kubuntu 13.04 Linux.

The Lazy Clause Generation [11] technology seems to be essential for our approach because our efforts to use another CP solver, Gecode 4.2.0 [15], failed even for 5-vessel subproblems. Packing problems are highly combinatorial, and this is where learning is the most advantageous. Moreover, some other LCG solvers than CPX did not work well, since this problem relies on lazy literal creation.

The solution of a MiniZinc model works in 2 phases. At first, it is *flattened*, i.e., translated into a simpler language FlatZinc. Then the actual solver is called on the flattened model. Time limits were imposed only on the second phase; in particular, we allowed at most 60 seconds in the EH heuristic and 30 seconds in an LNS iteration, see Section 2 for their details. However, reported times contain also the flattening which took a few seconds per model on average.

In EH and LNS, when writing the models with fixed subsets of the variables, we tried to omit as many irrelevant constraints as possible. In particular, this helped reduce the flattening time. For that, we imposed an upper bound of 200 hours on the maximal delay of any vessel (in the solutions, this bound was never achieved, see Figure 6 for example).

The default visibility horizon setting for our experiment, see Section 2.5, is VH 15/5: 15 vessels visible, they are approximately solved by EH and (possibly) improved by LNS(30, 15, 12); then the first 5 vessels are fixed, etc. Given the above time limits on an EH or LNS iteration, this takes less than 20 minutes to process each current visibility horizon and has shown to be usually much less because many LNS subproblems are proved infeasible rather quickly.

Our **test data** is the same as in [14]. It is historical data with compressed time to put extra pressure on the system. It has the following key properties:

– 358 vessels in the data file, sorted by their ETAs.
– One to three stockpiles per vessel, on average 1.4.
– The average interarrival time is 292 minutes.
– All ETAs are moved so that the first ETA = 10080 (7 days, to accommodate the longest build time).
– Optimizing vessel subsequences of 100 or up to 200 vessels, starting from vessels 1, 21, 41, . . . , 181.

Figure 6 illustrates the test data giving the delay profile in a solution for all 358 vessels. The solution is obtained with the default visibility horizon setting VH 15/5. The most difficult subsequences seem to be the vessel groups 1..100 and 200..270.

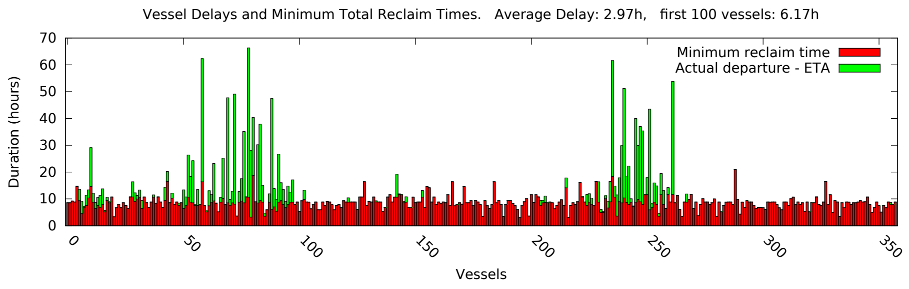


Fig. 6. Vessel delay profile in a solution of the instance 1..358.

Table 1. Solutions for the 100- and up to 200-vessel instances, obtained with EH, TTS, ALL, and VH 15/5.

	100 vessels								Up to 200 vessels							
	EH		TTS		ALL		VH 15/5		EH		TTS		ALL		VH 15/5	
1st	obj	*t*	obj	*t*	obj	*t*	obj	*t*	obj	*t*	obj	*t*	obj	*t*	obj	*t*
1	11.77	71	9.87	73	13.31	275	6.17	1509	6.15	170	5.09	90	7.06	356	3.19	1934
21	7.01	69	6.11	33	9.46	275	4.19	1758	3.75	142	3.25	68	5.08	352	2.23	2101
41	2.54	46	1.68	12	2.93	271	1.31	702	2.02	175	1.60	62	2.62	348	1.26	1465
61	0.64	42	0.61	18	0.98	273	0.51	214	3.59	252	3.25	60	5.39	351	2.63	1719
81	0.46	35	0.39	18	0.54	272	0.32	236	3.81	139	3.40	310	5.73	352	2.71	2084
101	0.33	29	0.23	7	0.52	272	0.19	202	3.39	140	3.21	62	5.14	352	1.91	2444
121	0.40	27	0.38	8	0.54	272	0.26	169	4.79	108	4.23	46	4.45	360	2.33	1815
141	2.82	154	1.44	20	2.59	273	1.35	612	4.72	220	3.26	47	4.45	353	2.45	2184
161	5.13	43	5.26	11	7.68	273	3.67	2031	3.53	101	3.26	42	5.15	352	2.25	2350
181	5.45	35	5.16	10	8.23	273	3.84	1438	3.13	70	2.93	33	4.72	328	2.17	1519
Mean	3.65	55	3.11	21	4.68	273	2.18	887	3.89	152	3.35	82	4.98	350	2.31	1961

4.1 Initial solutions

First we look at basic methods to obtain schedules for longer sequences of vessels. This is the EH heuristic from Section 2.3 and the TTS Greedy described in Section 3, which fit into the visibility horizon schemes VH 5/5 and VH 1/1, respectively. We compare them to an approach to construct schedules in a single MiniZinc model (method "ALL") and to the standard visibility horizon setting VH 15/5, Section 2.5. The results are given in Table 1 for the 100-vessel and 200-vessel instances.

Method "ALL", obtaining feasible solutions for the whole 100-vessel and 200-vessel instances in a single run of the solver, became possible after a modification of the default search strategy from Section 2.2. This did not produce better results however, so we present its results only as a motivation for iterative methods for initial construction and improvement.

The default solver search strategy proved best for the iterative methods EH and LNS. But feasible solutions of complete instances *in a single model* only appeared possible with a modification. Let us call the strategy from Section 2.2

LayerSearch(1,. . . ,|V|) because we start with all vessels' departure times, continue with reclaim times, pads, etc. The alternative strategy can be expressed as

$$\text{LayerSearch}(1,\ldots,5);\text{LayerSearch}(6,\ldots,10);\ldots$$

which means: search for departure times of vessels $1,\ldots,5$; then for the reclaim times of their stockpiles; then for their pad numbers; . . . departure times of vessels $6,\ldots,10$; etc. It is similar to the iterative heuristic EH with the difference that the solver has the complete model and (presumably) takes the first found feasible solution for every 5 vessels.

We had to increase the time limit per solver call: 4 minutes. But the flattening phase took longer than finding a first solution (there are a quadratic number of constraints). Feasible solutions were found in about 1–2 minutes after flattening. We also tried running the solver for longer but this did not lead to better results: the solver enumerates near the leaves of the search tree, which is not efficient in this case. Switching to the solver's default strategy after 300 seconds (search annotation `cpx_warm_start` [12]) gave better solutions, comparable with the EH heuristic.

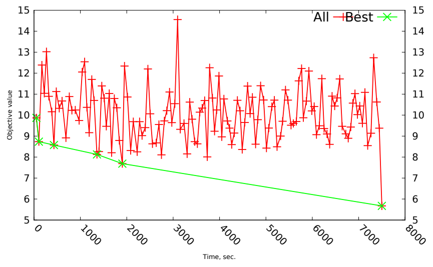
In Table 1 we see that the solutions obtained by the "ALL" method are inferior to EH. Thus, for all further tests we used strategy LayerSearch(1,. . . ,|V|) from Section 2.2. Further, EH is inferior to TTS, both in quality and running time. This proves the efficiency of the opportunity costs in TTS and suggests using TTS for initial solutions. However, TTS runs on original real-valued data and we could not use its solutions in LNS because the latter works on rounded data which usually has small constraint violations for TTS solutions. A workaround would be to use the rounded data in TTS but given the majority of running time spent in LNS, and for simplicity we stayed with EH to obtain starting solutions. The results for VH 15/5 where LNS worked on every visibility horizon, support this choice.

4.2 Visibility horizons

In this subsection, we look at the impact of varying the visibility horizon settings (Section 2.5), including the complete horizon (all vessels visible). More specifically, we compare $N = 1, 4, 6, 10, 15, 25$, or ∞ visible vessels and various numbers F of vessels to be fixed after the current horizon is scheduled. For $N = \infty$, we can apply a *global* solution method. Using Constraint Programming, we obtain an initial solution and try to improve it by LNS, denoted by *global LNS*, because it operates on the whole instance. Using Adaptive Greedy (Section 3), we also operate on complete schedules.

To illustrate the behaviour of global methods, we pick the difficult instance with vessels 1..100, cf. Figure 6. A graphical illustration of the progress over time of the global methods AG(130,0) and VH 15/5 + LNS(500, 15, 12) is given in Figure 7.

To investigate the value of various visibility horizons, for limited horizons, we applied the same settings as the standard one (Section 2.5): an initial schedule



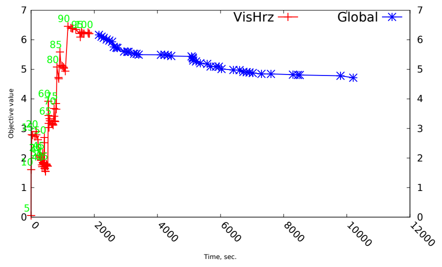


Fig. 7. Progress of the objective value in AG(130,0) (left) and VH 15/5 + LNS(500, 15, 12) (right), vessels 1..100

Table 2. Visibility horizon trade-off: all 100-vessel instances

N/F	1/1	4/2	6/3	10/5	15/7	15/1	25/12	25/5	**15/5+GLNS**
Delay, h	5.36	3.51	3.16	2.55	2.28	2.16	2.29	1.96	**1.73**
%Δ	210%	103%	83%	48%	32%	25%	33%	13%	
Time, s	114	188	202	267	916	2896	1823	3354	**4236**
%Δ	-97%	-96%	-95%	-94%	-78%	-32%	-57%	-21%	

for the current horizon is obtained with EH and then improved with LNS(30, 15, 12). Table 2 gives the average results for all 100-vessel instances. On average, the global Constraint Programming approach (500 LNS iterations) gives the best results, but VH 25/5 is close. Moreover, the setting VH 15/1 which invests significant effort by fixing only one vessel in a horizon, is slightly better than VH 25/12, which shows that with a smaller horizon, more computational effort can be fruitful.

The visibility horizon setting 1/1 produces the worst solutions. The TTS heuristic of Section 3 also uses this visibility horizon, but produces better results, see Table 3. The reason is probably the more sophisticated search strategy in TTS, which minimizes 'opportunity costs' related to reclaimer flexibility. At present, it is impossible to implement this complex search strategy in MiniZinc, the search sublanguage would need significant extension to do so.

4.3 Comparison of Constraint Programming and Adaptive Greedy

To compare the Constraint Programming and the AG approaches, we select the following methods: **VH 15/5** Visibility horizon 15/5; **VH 15/5+G** Visibility horizon 15/5, followed by global LNS 500; **VH 25/5** Visibility horizon 25/5; $\mathbf{AG}_1$ TTS Greedy, one iteration on the ETA order; $\mathbf{AG}_{500/500}$ Adaptive greedy, 500 iterations in both phases; $\mathbf{AG}_{1000/0}$ Adaptive greedy, 1000 iterations in Phase I only. The results for the 100-vessel instances are in Table 3. The pure-random configuration of the Adaptive Greedy, $\mathrm{AG}_{0/1000}$, showed inferior performance, and its results are not given.

Table 3. 100 vessels: VH and LNS vs. (adaptive) greedy

	Constraint Programming							TTS Greedy and Adaptive Greedy							
	VH 15/5		VH 15/5*+G			VH 25/5		AG_1		$AG_{500/500}$			$AG_{1000/0}$		
1st	obj	*t*	obj	*t*	iter	obj	*t*	obj	*t*	obj	*t*	iter	obj	*t*	iter
1	6.17	1509	4.72	10204	338	5.44	7507	9.87	73	5.67	9992	130	5.67	7500	130
21	4.19	1758	3.17	10529	494	3.30	6626	6.11	33	3.63	10433	333	3.25	23051	991
41	1.31	702	1.24	922	0	1.24	3256	1.68	12	1.00	11253	667	1.02	12660	903
61	0.51	214	0.50	279	0	0.51	912	0.61	18	0.54	2713	126	0.54	2769	126
81	0.32	236	0.32	299	0	0.32	1266	0.39	18	0.34	11972	696	0.36	5794	344
101	0.19	202	0.19	285	2	0.18	943	0.23	7	0.21	7764	771	0.22	15	1
121	0.26	169	0.26	258	3	0.26	971	0.38	8	0.28	3589	525	0.29	201	28
141	1.35	612	0.73	4883	469	0.90	1364	1.44	20	0.80	4895	255	0.76	12969	652
161	3.67	2031	2.50	8172	369	3.51	4241	5.26	11	3.24	12574	845	3.78	4166	284
181	3.84	1438	3.64	6525	311	3.89	6450	5.16	10	3.83	6818	422	3.65	13843	809
Mean	2.18	887	**1.73**	4236	199	1.96	3354	3.11	21	**1.95**	8200	477	**1.95**	8297	427

* For limited visibility horizons, LNS(20,12,12) was applied

5 Conclusions

We consider a complex problem involving scheduling and allocation of cargo assembly in a stockyard, loading of cargoes onto vessels, and vessel scheduling. We designed a Constraint Programming (CP) approach to construct feasible solutions and improve them by Large Neighbourhood Search (LNS).

Investigation of various visibility horizon settings has shown that larger numbers of known arriving vessels lead to better results. In particular, the visibility horizon of 25 vessels provides solutions close to the best found. The new approach was compared to an existing greedy heuristic. The latter works with a visibility horizon of one vessel and, under this setting, produces better feasible solutions in less time. The reason is probably the sophisticated search strategy which cannot be implemented in the chosen CP approach at the moment. To make the comparison fairer, an adaptive iterative scheme was proposed for this greedy heuristic, which resulted in a similar performance to LNS.

Overall the CP approach using visibility horizons and LNS generated the best overall solutions in less time than the adaptive greedy approach. A significant advantage of the CP approach is that it is easy to include additional constraints, which we have done in work not reported here for space reasons.

Acknowledgments The research presented here is supported by ARC linkage grant LP110200524. We would like to thank the strategic planning team at HVCCC for many insightful and helpful suggestions, Andreas Schutt for hints on efficient modelling in MiniZinc, as well as to Opturion for providing their version of the CPX solver under an academic license.

Bibliography

[1] Bay, M., Crama, Y., Langer, Y., Rigo, P.: Space and time allocation in a shipyard assembly hall. Annals of Operations Research 179(1), 57–76 (2010)

[2] Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling 20(12), 97–123 (1994)

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

[4] Boland, N.L., Savelsbergh, M.W.P.: Optimizing the Hunter Valley Coal Chain. In: Gurnani, H., Mehrotra, A., Ray, S. (eds.) Supply Chain Disruptions, pp. 275–302. Springer London (2012)

[5] Clautiaux, F., Jouglet, A., Carlier, J., Moukrim, A.: A new constraint programming approach for the orthogonal packing problem. Computers & Operations Research 35(3), 944–959 (2008)

[6] HVCCC: Hunter valley coal chain — overview presentation (2013), http://www.hvccc.com.au/

[7] Lesh, N., Mitzenmacher, M.: BubbleSearch: A simple heuristic for improving priority-based greedy algorithms. Information Processing Letters 97(4), 161–169 (2006)

[8] Lodi, A., Martello, S., Vigo, D.: Recent advances on two-dimensional bin packing problems. Discrete Applied Mathematics 123(1–3), 379–396 (2002)

[9] Marriott, K., Stuckey, P.J.: A MiniZinc tutorial (2012), http://www.minizinc.org/

[10] Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation = lazy clause generation. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming — CP 2007, Lecture Notes in Computer Science, vol. 4741, pp. 544–558. Springer Berlin Heidelberg (2007)

[11] Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)

[12] Opturion Pty Ltd: Opturion CPX user's guide: version 1.0.2 (2013), www.opturion.com

[13] Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, pp. 399–419. Springer US (2010)

[14] Savelsbergh, M., Smith, O.: Cargo assembly planning. Tech. rep., University of Newcastle (2013), accepted

[15] Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with Gecode (2013), www.gecode.org

[16] Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (2011)

[17] Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. Journal of Scheduling 16(3), 273–289 (2013)

[18] Schutt, A., Stuckey, P.J., Verden, A.R.: Optimal carpet cutting. In: Lee, J. (ed.) Principles and Practice of Constraint Programming — CP 2011, Lecture Notes in Computer Science, vol. 6876, pp. 69–84. Springer Berlin Heidelberg (2011)

[19] Singh, G., Sier, D., Ernst, A.T., Gavriliouk, O., Oyston, R., Giles, T., Welgama, P.: 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 (2012)

[20] Thomas, A., Singh, G., Krishnamoorthy, M., Venkateswaran, J.: Distributed optimisation method for multi-resource constrained scheduling in coal supply chains. International Journal of Production Research 51(9), 2740–2759 (2013)