EURO J Transp Logist (2012) 1:197–236
DOI 10.1007/s13676-012-0011-z

RESEARCH PAPER

A stockyard planning problem

Natashia Boland · Damon Gulczynski · Martin Savelsbergh

Received: 16 October 2011 / Accepted: 13 May 2012 / Published online: 20 June 2012

Abstract With over 100 millions tons of coal exported annually, the coal terminals at the Port of Newcastle are among the busiest in the world. To accommodate the anticipated rise in the export of coal in the next decade, effective stockyard management at the coal terminals will be essential. We have developed stockyard planning technology that can achieve substantially higher throughput levels. The technology intelligently combines greedy construction, enumeration, and integer programming. We demonstrate the efficacy of the technology on a variety of instances derived from real-life data and also show how the technology can be used to investigate the benefits of capacity expansion investments on throughput levels.

Keywords Coal export chain · Stockyard management · Integer programming · Heuristic

Introduction

As the world population grows and the living standards in the developing world improve, demand for electricity increases. For the foreseeable future, increased demand for electricity translates to increased demand for coal, the prevalent energy source for power generation. Coal is also vital for steel-making, and steel mills across Asia, especially in China, are experiencing increases in demand, further increasing the demand for coal. For the sake of completeness, we mention that there are varying opinions regarding the increased coal use and its effect on world development, see for example, Muller et al. (2011).

N. Boland · D. Gulczynski · M. Savelsbergh (✉)
School of Mathematical and Physical Sciences, University of Newcastle, Callaghan, NSW 2308, Australia
e-mail: martin.savelsbergh@newcastle.edu.au

With over 100 million tons of coal exported annually, the Hunter Valley Coal Chain (HVCC) is the largest exporter of coal in the world. Thirteen coal producers owning approximately 35 mines operate in the Hunter Valley region of New South Wales, Australia. Coal is exported through three coal terminals at the Port of Newcastle servicing more than 1,000 coal-carrying ships annually. The Hunter Valley Coal Chain Coordinator (HVCCC) is responsible for the coordination and planning of the coal transportation process, a highly complex logistical operation that begins with rail scheduling at mine load points and ends with the load planning of ships at the port. For a more in-depth description of the HVCC see Boland and Savelsbergh (2012).

In this paper, we focus mainly on the terminal end of the coal export chain, but we cannot completely ignore the front end of the chain, as operations at the terminals are heavily dependent on a rail system transporting coal from mine load points to the terminals. Primarily, we look at the stockyards in the coal terminals. The stockyards are where cargoes of (typically blended) coal product are assembled in stockpiles using stacking machines, and then reclaimed using bucket wheel reclaimers for transport via conveyor belts to shiploaders at the berths.

An important characteristic of a coal terminal is whether it operates as a cargo assembly terminal or as a dedicated stockpiling terminal. When a terminal operates as a cargo assembly terminal, it operates in a "pull-based" manner, where the coal blends assembled and stockpiled at the terminal are based on the demands of the arriving ships. When a terminal operates as dedicated stockpiling terminal, it operates in a "push-based" manner, where a small number of coal blends are built in dedicated stockpiles and only these coal blends can be requested by arriving vessels. We focus on cargo assembly terminals as they are more difficult to operate due to the large variety of coal blends that needs to be accommodated.

At a cargo assembly terminal the planning process can be thought of, conceptually, as consisting of four phases. In the first phase, the ships arriving at the port are assigned to one of the berths at the terminal. In the second phase, the cargoes (blends of coal) that need to be assembled and loaded onto the vessel are allocated a space in the stockyard. In the third phase, a cargo assembly plan is developed that indicates the daily movements of coal from the mines to the stockyard. Finally, in the fourth phase, a detailed cargo assembly schedule is produced that specifies the precise timing of the trains that bring coal from the mines to the terminal.

Depending on the size and the coal blend of a cargo, the assembly may take anywhere from two to ten days. This is, in part, due to the fact that some of the mines are located several hundred miles away from the port and just getting a single trainload of coal to the port takes a considerable amount of time. We note that this aspect of the operation will become increasingly important as mines to be opened in the future will be further away from the port.

Once coal has been delivered to the stockyard, and the assembly of a stockpile has started, it is rare that the location of the stockpile in the stockyard is changed; relocating a stockpile is time-consuming and requires resources (stackers and reclaimers) that could be allocated to other stockpiles. Thus, decisions on where to locate a stockpile and when to start the assembly of a stockpile are critical for the

overall efficiency of the system. The amount of time it takes to assemble a stockpile depends on the movements of coal from the mines to the stockyard, and thus it depends on the available load point capacity at the mines and stacking capacity at the stockyard. Because of this, such capacities need to be taken into account when making decisions regarding the location and assembly start time of stockpiles. This is the underlying reason for having a cargo assembly plan.

In this paper, we discuss the development of stockyard planning technology that considers the first three phases of the planning process simultaneously. Thus, we focus on berth allocation decisions, stockpile location decisions, and stockpile assembly start time decisions. Since we consider a stream of ship arrivals when making these decisions, we also decide on reclaiming start times. Ideally, the assembly of the stockpiles for a ship completes at the time the ship arrives at the berth (i.e., "just-in-time" assembly) and the reclaiming of the stockpiles commences immediately. Unfortunately, this may not be possible due to the capacities of the resources in the system, e.g., load points, trains, stackers, stockyard space, and reclaimers.

Another complication that cannot be ignored is the fact that the Port of Newcastle is a tidal-constrained river port. Consequently, the largest ships, i.e., cape-class vessels which can carry up to 180,000 tons of coal, can only leave the port during high tide. At the most three cape-class vessels can leave the port during a period of high tide.

The throughput capacity of the coal export chain, i.e., the maximum number of tons of coal per year that can be exported, is of crucial importance considering the anticipated growth in demand for coal. However, determining the throughput capacity is an extremely complex task due to the many interacting components that make up the coal export chain. The HVCCC is continuously looking for ways to increase the throughput capacity even without knowing what the current throughput capacity is. The throughput capacity is strongly correlated with the maximum number of ships that can be accommodated per year. Therefore, our technology aims to minimize the departure time of the last ship for a given sequence of ship arrivals at the port. However, since minimizing the departure time of the last ship is an awkward objective to work with, we use minimizing the mean delay of ships as a proxy for that objective, where the delay of a ship is the difference between the ship's departure time and its earliest possible departure time (i.e., the departure time in a system with infinite capacity).

At the heart of our technology is a hybrid algorithm that combines ideas from greedy construction heuristics, enumeration, and integer programming. It strikes a delicate balance between efficiency and quality. As it is expected that this type of technology may be used several times a day to assess the impact of new nominations, i.e., information regarding additional arriving ships, efficiency is an important consideration. Compared to a baseline greedy construction heuristic which mimics, to some extent, the current process followed by the planners responsible for stockyard management, the hybrid algorithm is able to reduce the mean delay an order of magnitude and increase the throughput significantly. Furthermore, a computational study analyzing the impact of capacity expansions, from increasing load point capacity to increasing stockyard space, has revealed, not

surprisingly, that the impact of capacity expansions varies dramatically and that therefore a thorough analysis is required before any capital investments are made.

The stockyard planning problem

The stockyard planning problem to be defined below is motivated by the planning process for one of the cargo assembly terminals at the Port of Newcastle, which is assumed to be representative of the planning process for any cargo assembly terminal.

At this particular terminal, there are four pads in the stockyard on which cargoes are assembled. Coal is transported from load points associated with one or more mines to the stockyard at the terminal by train. Upon arrival at the stockyard, a train dumps its contents at one of three dump stations. The coal is then transported on a conveyor to a pad where it is added to a stockpile by a stacker. There are six stackers. Each stockpile of coal spans the entire width of a pad, so we can think of stockpiles and pads as effectively being one-dimensional. A single stockpile is usually built from several train loads over several days. After a stockpile is completely built, it usually must dwell on its pad for some time (perhaps several days) until the ship onto which it is to be loaded is available at a berth. A stockpile is reclaimed using a bucket-wheel reclaimer and the coal transferred to a berth on a conveyor. It is then loaded onto a ship with a shiploader.

Figure 1 shows part of the layout of the terminal. The terminal has three dump stations, four pads, six stackers, four reclaimers, four berths, and three shiploaders. The four pads (Pad A–Pad D) are oriented so that their easternmost edges are aligned at a position we have designated as position 0. Pads A and B are 2,450 m long. Pads C and D are 1,400 m long (these pads are expected to be extended to 2,450 m by January 2012). The three dump stations are located just north of the pads. Two stackers service Pad A, two stackers service Pads B and C, and two stackers service Pfad D. Two reclaimers service Pads A and B, and two reclaimers service Pads C and D. Stackers and reclaimers can move back and forth in their slots to service stockpiles on different positions of a pad. However, machines in the same slot must remain at least 30 m from one another at all times, and they cannot "pass" each other. The three ship loaders and the four berths are located on the waterfront just south of the pads.

A brief overview of some of the key events driving the planning process is presented next. An incoming ship alerts the coal chain managers that it will be arriving some time before its estimated time of arrival (*ETA*). This announcement is referred to as the ship's nomination. Upon nomination, a ship provides a specification of the cargoes to be assembled to the coal chain managers. As coal is a blended product the specification includes for each cargo a recipe indicating from which mines coal needs to be sourced and in what quantities. At this time, the assembly of the cargoes (stockpiles) for the ship can begin. In practice, most ships request only one or two cargoes, but there are ships that request as many as six cargoes. A ship cannot arrive at a berth prior to its *ETA*, and often a ship has to wait until after its *ETA* for a berth to become available. Once at a berth, and once all its

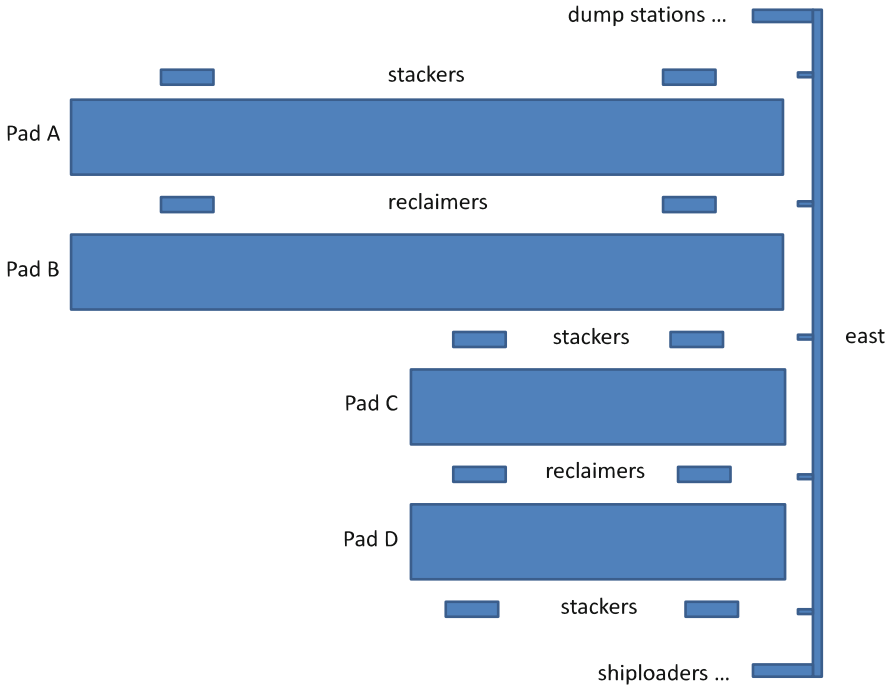

Fig. 1 Partial layout of the terminal

cargoes have been assembled, the reclaiming of the stockpiles (the loading of the ship) can begin. A ship must be loaded in a way that maintains its physical balance in the water. As a consequence, for ships with multiple cargoes, there is a predetermined sequence in which its cargoes must be reclaimed. If a ship is particularly large (it can hold more than 100,000 tons of coal), it can only leave during certain time periods based on the high tides. At most three such ships can depart during a period of high tide, and in that case the departures must be 90 min prior to the high tide, 30 min prior to the high tide, and 30 min after the high tide. Furthermore, at the Port of Newcastle, it is not possible for two large ships to depart at the same time, because the channel is not wide enough to accommodate more than one large ship at a time.

The goal of the planning process is to maximize the throughput without causing unacceptable delays for the ships. We focus on berth allocation decisions, stockpile location decisions, cargo assembly plan decisions, and stockpile reclaiming decisions. These decisions are typically made two to three weeks before a ship's expected arrival at the port. We do not decide on a detailed cargo assembly schedule specifying the precise timing of the trains that bring coal from the mines to the terminal. These operational decisions are typically made the day before the coal movements take place. Therefore, cargo assembly plan decisions, i.e., decisions regarding the coal movements from load points to the stockyard, are modeled at an aggregate daily granularity, whereas berth allocation and reclaiming decisions are

modeled at a much finer, essentially continuous, granularity. As the coal movements are modeled at a daily granularity, there is no need to explicitly model dump stations and to distinguish between the stackers in the same slot. It suffices to consider three "stacker streams" and account for dump station and stacker capacity by means of a daily stacker stream capacity. Also, we do not explicitly model ship loaders. We assume that a stockpile is reclaimed directly onto a ship. Consequently, we use the terms "reclaim" and "load" synonymously. Since there are four reclaimers, but only three ship loaders, we have to ensure that only three ships are being loaded at any given time.

The SPP captures the characteristics of the decision environment outlined above. For a given set of ships arriving at the port, the goal is to assign each ship to a berth and schedule its arrival time at that berth, to assign each cargo of the ship to a location in the stockyard and schedule the coal movements required to assemble that cargo, to schedule the reclaiming of the cargoes, and to schedule the ships's departure, so as to minimize the average delay of the ships. The delay of a ship is defined to be the difference between the scheduled departure time of a ship and its baseline time. A ship's baseline time is the earliest time the ship could depart under ideal circumstances, i.e., it is the departure time that results if we assume the ship arrives at the berth at its *ETA* and all its stockpiles are ready to be reclaimed immediately upon its arrival. For large ships, the baseline time is adjusted, if necessary, to take the high tide into account.

In the SPP, we have to ensure that no two ships occupy the same berth at the same time. Furthermore, to avoid conflicts between departing and arriving ships at a berth, a certain amount of time (approximately 1 h, in practice) must elapse after a ship departs a berth before the next ship can arrive. We have to ensure a minimum distance (10 m, in practice) between adjacent stockpiles on a pad to avoid contamination of the stockpiles. We have to ensure that the building of a stockpile does not start before a ship's nomination. That is, the arrival of the first train with coal for the stockpile cannot occur before a ship's nomination. The total tonnage from a particular load point required for a stockpile is known in advance (that information can be derived from the size of the cargo and the recipe of the cargo) as is the capacity of the train type that serves that load point. Thus, we can calculate the number of coal movements required for a stockpile from a load point. We assume that the total tonnage is evenly distributed over these coal movements. Furthermore, we assume that a train arrives at the terminal at 12:00 a.m. on the day the coal movement is scheduled to take place. When scheduling coal movements, we must respect daily capacities at the load points. Each load point has a maximum daily tonnage that it can produce and a maximum number of coal movements that it can accommodate per day. Daily stacker stream capacities also have to be respected. Each stacker stream has a maximum number of productive hours per day. The rate at which a stacker stream can process an arriving train depends on the originating load point. That rate is used to calculate the total time it takes to stack the coal from the train on the stockpile. A given reclaimer can only service stockpiles on certain pads (see Fig. 1). A stockpile must be completely built before it can be reclaimed. If the last train with coal for a

stockpile arrives at 12:00 a.m. on a given day, then we assume that the stockpile is not completely built until 12:00 a.m. the next day. The stockpiles of a ship must be reclaimed in a predetermined order, and we cannot start reclaiming a stockpile until the stockpile before it in the order has been completely reclaimed. Furthermore, we cannot start reclaiming the first stockpile of a ship until *all* stockpiles of that ship have been completely built. We assume the rate of reclaiming is the same for all reclaimers, and thus the time it takes to reclaim a given stockpile can be calculated from its size. We have to account for the transit time of a reclaimer when it moves from one stockpile to the next. This transit time depends on the speed of the reclaimer (30 m per minute for all reclaimers) and the positions of the two stockpiles on their pads. We have to avoid reclaimer conflicts. Specifically, if reclaimer A is located behind reclaimer B in a given slot, then at any given time, the position of reclaimer A must be no greater than the position of reclaimer B minus a buffer distance (30 m). There is a maximum number of reclaimers (three) that can be in use at any given time. In other words, there is a limit as to how many ships can be at berth being loaded at one time. A ship can depart after its last stockpile has been reclaimed plus a certain amount of time to allow for paperwork processing. Large ships can only depart 90 min prior to, 30 min prior to, or 90 min after a high tide. No two large ships can depart at the same time.

Next, we provide a detailed description of the SPP, in which we formalize the many complexities of the stockyard system discussed above. Furthermore, we introduce the notation that will be used throughout the remainder of the paper. We use the convention of denoting parameters with three letters and decisions with one letter; with one exception: the planning horizon is denoted by T. Superscripts give a description of the parameter or decision and subscripts are indices. To add precision to the description, we often present mathematical expressions. Together, these expressions are not intended to, and do not, provide a mathematical programming formulation of the problem.

Parameter sets

V	Set of ships
V^L	Set of large ships
S	Set of stockpiles
B	Set of berths
P	Set of pads
$H(p)$	Set of positions on pad p
K	Set of stacker streams
$K(p)$	Set of stacker streams that service pad p
L	Set of load points
R	Set of reclaimers
$R(p)$	Set of reclaimers that service pad p
$\hat{T}$	Set of times representing the start of a day
T^L	Set of times at which a large ship can depart

$S(v)$	Set of stockpiles of ship v, given in the order in which they must be reclaimed ($S(v) = \{s_{v1}, s_{v2},..., s_{vn_v}\}$, where n_v is the number of stockpiles of v, and s_{vi} is the ith stockpile in v's reclaiming sequence)
$S(l)$	Set of stockpiles that require at least one coal movement from load point l
$L(s)$	Set of load points from which stockpile s requires at least one coal movement

Parameters

nom_v	Nomination time of ship v
eta_v	Estimated time of arrival of ship v
buf_v^{atl}	Amount of time that must elapse after ship v arrives at a berth before loading can begin
buf_v^{dta}	Amount of time that must elapse after ship v departs a berth before the next ship can arrive
buf_v^{dtd}	Amount of time that must elapse after ship v has been loaded, before it can depart
len_s	Length of stockpile s on a pad
dur_s^{rec}	Amount of time it takes to reclaim stockpile s
dur_v^{rec}	Amount of time it takes to reclaim all stockpiles of ship v ($\sum_{s \in V(s)} dur_s^{rec}$)
bsl_v	Baseline time of ship v (time at which delay starts to incur, for non-large ships $bsl_v = eta_v + buf_v^{atl} + dur_v^{rec} + buf_v^{dtd}$, for large ships $bsl_v = \min\{t \in T^L: t \geq eta_v + buf_v^{atl} + dur_v^{rec} + buf_v^{dtd}\}$)
num_{sl}^{mov}	Total number of coal movements that must be carried out for stockpile s from load point l
num_v^{mov}	Total number of coal movements that must be carried for ship v
lod_{sl}^{ton}	Load (tonnage) of a coal movement for stockpile s from load point l
dur_{slk}^{stk}	Amount of time it takes to stack a train load for stockpile s from load point l using stacker stream k
len_p	Length of pad p
cap_k^{dur}	Daily capacity, in terms of working hours, of stacker stream k
cap_l^{mov}	Number of coal movements that can be accommodated at load point l in one day
cap_l^{ton}	Daily capacity, in terms of tonnage, of load point l
siz_l^{trn}	Size (in tons) of the trains that service load point l
buf_p^{dis}	Buffer distance that must be maintained between adjacent stockpiles on pad p
$dur_{rhh'}$	Amount of time that elapses when reclaimer r moves from position h to position h'
$buf_{rr'}^{dis}$	Buffer distance that must be maintained at all times between reclaimers r and r' in the same slot

num^{mxr} Maximum number of reclaimers that can be in use at one time (number of ship loaders)

Decisions

b_v Berth to which ship v is assigned
t_v^{arr} Time of arrival of ship v at berth b_v
p_s Pad on which stockpile s is assembled
h_s Position of stockpile s on its pad (position of the easternmost edge of s)
r_s Reclaimer used in reclaiming stockpile s
n_{slt}^{mov} Number of coal movements carried out for stockpile s from load point l at time $t \in \hat{T}$
t_s^{beg} Time at which reclaiming of stockpile s starts

Implied decisions

t_v^{dep} Time of departure of ship v from berth b_v
h_s^{mid} Position of the middle of stockpile s on its pad ($h_s^{mid} = h_s + \frac{len_s}{2}$)
t_s^{fst} Time at which building of stockpile s starts (the day the first coal movement for s takes place; $t_s^{fst} = \min \{t \in \hat{T} : \exists\, l \in L(s)$ such that $n_{slt}^{mov} > 0\}$)
t_s^{lst} Time at which building of stockpile s finishes (one day after the last coal movement for s takes place; $t_s^{lst} = \max \{t \in \hat{T} : \exists\, l \in L(s)$ such that $n_{slt}^{mov} > 0\}$ + one day)
t_s^{end} Time at which reclaiming of stockpile s finishes ($t_s^{end} = t_s^{beg} + dur_s^{rec}$)
t_v^{end} Time at which reclaiming of the last stockpile of ship v finishes ($t_v^{end} = t_{s_{n_v}}^{end}$)

Implied sets

$V(b)$ Set of ships scheduled to arrive at berth b
$S(k)$ Set of stockpiles located on a pad serviced by stacker stream k
$S(r)$ Set of stockpiles reclaimed by reclaimer r
$S(p, t)$ Set of stockpiles on pad p at time t
$R(t)$ Set of reclaimers in use at time t

Objective

Minimize the mean delay of ships:

$$\frac{1}{|V|}\sum_{v \in V}(t_v^{dep} - bsl_v)$$

Constraints

- A ship cannot arrive at a berth before its *ETA*:

$$eta_v \leq t_v^{arr} \quad \forall v \in V$$

- No two ships can occupy the same berth at the same time:

$$t_v^{dep} + buf_v^{dta} \leq t_{v'}^{arr} \quad \forall b \in B, \forall v, v' \in V(b) \text{ such that } t_{v'}^{arr} \geq t_v^{arr}$$

- A stockpile must fit on its pad:

$$h_s + len_s \leq len_p \quad \forall s \in S$$

- A stockpile cannot conflict with another stockpile on a pad:

$$h_s + len_s + buf_p^{dis} \leq h_{s'} \quad \forall p \in P, \forall t \in [0, T], \forall s, s' \in S(p, t) \text{ such that } h_{s'} \geq h_s$$

- Building of a stockpile cannot begin before its ship's nomination:

$$t_s^{fst} \geq nom_v \quad \forall v \in V, \forall s \in S(v)$$

- All coal movements for a stockpile from a load point must be carried out:

$$\sum_{t \in \hat{T}} n_{slt}^{mov} = num_{sl}^{mov} \quad \forall s \in S, \forall l \in L(s)$$

- The total number of coal movements from a load point on a given day cannot exceed its capacity:

$$\sum_{s \in S(l)} n_{slt}^{mov} \leq cap_l^{mov} \quad \forall l \in L, \forall t \in \hat{T}$$

- The total tonnage delivered from a load point on a given day cannot exceed its capacity:

$$\sum_{s \in S(l)} lod_{sl}^{ton} n_{slt}^{mov} \leq cap_l^{ton} \quad \forall l \in L, \forall t \in \hat{T}$$

- The total working hours of a stacker stream on a given day cannot exceed its capacity:

$$\sum_{s \in S(k)} \sum_{l \in L(s)} dur_{slk}^{stk} n_{slt}^{mov} \leq cap_k^{dur} \quad \forall k \in K, \forall t \in \hat{T}$$

- Reclaiming of the first stockpile of a ship cannot start until all the ships stockpiles have been built:

$$t_{s_{v1}}^{beg} \geq t_s^{lst} \quad \forall v \in V, \forall s \in S(v)$$

- Stockpiles of a ship must be reclaimed in order:

$$t_{s_{v(i+1)}}^{beg} \geq t_{s_{vi}}^{beg} \quad \forall v \in V, \forall s_{vi}, s_{v(i+1)} \in S(v)$$

- A stockpile cannot be reclaimed by a reclaimer that does not service its pad:

$$r_s \neq r \quad \forall s \in S, \forall r \notin R(p_s)$$

- A reclaimer cannot work on two stockpiles at the same time:

$$t_s^{end} + dur_{rh_s^{mid}h_{s'}^{mid}} \leq t_{s'}^{beg} \quad \forall s, s' \in S(r) \text{ such that } t_{s'}^{beg} \geq t_s^{beg}$$

- Reclaimers in the same slot must maintain a buffer distance between them:

$$h_s + len_s + buf_{rr'}^{dis} \leq h_{s'} \quad \forall r, r' \in R \text{ such that } r \text{ is behind } r' \text{ in the same slot, and}$$
$$\exists\, s \in S(r), s' \in S(r') \text{ such that } t_s^{beg} \leq t \leq t_s^{end} \text{ and } t_{s'}^{beg} \leq t \leq t_{s'}^{end}$$

- The number of reclaimers in use at any given time cannot exceed the maximum:

$$|R(t)| \leq num^{mxr} \quad \forall t \in [0, T]$$

- A ship cannot depart until all its stockpiles have been reclaimed:

$$t_v^{dep} \geq t_{s_{n_v}}^{end} + buf_v^{ltd} \quad \forall b \in B, \forall v \in V(b)$$

- A large ship can only depart at designated times based on high tides:

$$t_v^{dep} = h \quad \forall v \in V^L, \text{ for some } h \in T^L$$

- No two large ships can depart at the same time:

$$t_v^{dep} \neq t_{v'}^{dep} \quad \forall v, v' \in V^L$$

As mentioned above, the mathematical expressions are there only to make the descriptions more precise, they do not comprise a mathematical programming formulation. No doubt a mathematical programming formulation can be produced, but it is not likely to be of immediate use for solving the problem. Locating stockpiles on a pad, for example, resembles 2D strip packing, which is notoriously hard for mathematical programming approaches due to the inherent non-linearities. Locating stockpiles on the pad is more complicated than 2D strip packing because the length of one side of the space–time rectangles, the side corresponding to time, is not fully known as it depends on other decisions, namely when coal movements are scheduled). Also, because reclaimers serving the same pads cannot get too close to each other and cannot pass each other, reclaimer scheduling involves both space and time as well, again leading to non-linearities in the formulation. To be able to use powerful and robust commercial integer programming solvers such as CPLEX, Gurobi, or XPRESS, linearization techniques may be used, but they will either involve "big-M" constructs or a discretization of time, both of which will lead to formulations that are not likely to lead to satisfactory results, from a computing time perspective, for the real-life instances of interest. This is one of the reasons for initially pursuing heuristic approaches.

In Figure 2, we show an example time line for a ship v with a single stockpile s. For ship v, we show its nomination time and the times at which it is at berth. For

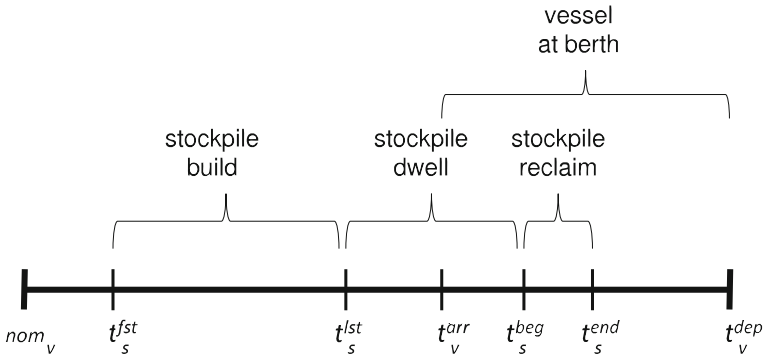

Fig. 2 Example time line of a ship with a single stockpile

stockpile s, we show the times at which it is being built (coal movements are taking place), the times at which it is dwelling idly on the pad, and the times at which it is being reclaimed.

Related literature

To the best of our knowledge, there are few problems discussed in the literature that are similar to the SPP. Certain aspects of the SPP are well-studied, e.g., scheduling of trains and optimizing storage at seaport terminals, but few, if any, papers consider these aspects simultaneously in a single comprehensive problem setting. The problem described by Abdekhodaee et al. (2004) probably comes closest to the SPP. They also consider a coal chain in Australia, but focus more on train scheduling than on stockyard management. They decompose the problem into several modules and solve each of the modules separately using a greedy heuristic. Conradie et al. (2006) consider optimizing the supply of coal to a factory producing liquid fuel products. Their focus is on developing robust schedules that can accommodate fluctuations in the product demand observed at the factory. A simulated annealing approach capable of generating near-optimal schedules is presented.

Other than the work by Abdekhodaee et al. (2004) and Conradie et al. (2006), we did not find papers concerned with the optimization of mineral supply chains from mine to port or from mine to plant or factory. There are, however, a number of simulation studies of mineral supply chains. For example, Glassrock and Hoare (1995) describe a discrete-event simulation system of an iron-ore supply chain, and, more recently, Welgama and Oyston (2003) present a discrete-event simulation system of the HVCC. These simulation systems are built and used to support strategic planning activities—simulations are run to evaluate the effects of potential changes in the supply chain.

Next, we focus on components and aspects of the coal export chain. Newman et al. (2010) review applications of operations research in mine planning. Liu and Kozan (2009) focus on the movement of coal in the coal export chain and model the scheduling of coal-carrying trains as a blocking parallel-machine job shop scheduling problem. Many researchers have considered the optimal storage of containers in a yard. For example, Lee et al. (2009) develop a hybrid insertion algorithm, Wong and Kozan (2010) design a tabu search algorithm, and Ng et al. (2010) build an iterative clustering algorithm. Recently, Cordeau et al. (2011) implemented an adaptive large neighborhood search algorithm for optimizing yard assignments in an automotive transhipment terminal.

The SPP has elements of a 2D packing problem because a stockpile takes up space and time on a pad in the stockyard, and no two stockpiles can occupy the same space at the same time on a pad. Therefore, when locating stockpiles in the stockyard, we must consider a space and a time dimension, like in a 2D packing problem. Over the years, there has been extensive work done on the 2D packing problem, see Lodi et al. (2002) for a survey. However, the problem of locating stockpiles in the stockyard in the SPP is more complex than a traditional 2D packing problem, because the time dimension of a stockpile "rectangle" is not known in advance and depends on the coal movement and reclaiming schedule of the stockpile. Furthermore, the objective is not to minimize wasted space, but to minimize average ship delay, which depends on the time dimensions of the rectangles. In a recent paper, Bay et al. (2010) model a complex packing problem with elements that resemble those of our stockpile management setting. In their problem, large building blocks for manufacturing ships must be located efficiently in an assembly hall. The hall is divided into equal-sized rectangular areas, with blocks to be positioned in each area for the duration of their assembly. In this way, the hall is similar to the stockyard—an area resembles a pad and a block resembles a stockpile. A major difference between the models, however, is that the time dimensions of the blocks are fixed and known in advance, unlike the time dimension of the stockpiles.

Algorithms

In this section, we motivate and describe three progressively more sophisticated algorithms for solving the SPP. The initial algorithm is a greedy construction algorithm that mimics, to some extent, the process currently followed by the human planners. This greedy construction algorithm is subsequently improved by changing some of the decision rules and by replacing greedy decision rules by optimization models.

We are taking a pragmatic approach here. We are dealing with a large-scale, complex, real-life decision problem and to assess the potential benefits of

optimization in that environment we have chosen to initially develop heuristic approaches.

An *ETA*-based greedy construction algorithm

The basic *ETA*-based greedy construction algorithm (GCA-ETA) schedules ships (and their cargoes) one at a time in non-decreasing order of their *ETA*s. This involves for each vessel the following steps:

- Assign ship v to the first available berth at or after the ship's *ETA*. In Fig. 3, we give an example with four berths (B1—B4) and eight ships (V1–V8) whose arrivals and departures have already been scheduled. Time is presented on the x-axis, so that the width of a rectangle represents the period during which a ship is at berth. The ship currently being process, V9, will be assigned to berth B1 with its scheduled arrival time set to be the departure time of ship V6 plus the minimum amount of time that must elapse between the departure of a ship from a berth and the arrival of the next ship at the berth.
- For each stockpile $s \in S(v)$, in the order in which the stockpiles need to be reclaimed, locate s in the stockyard in a way such that the first coal movement for s can be carried out as early as possible. Only locations at the beginning of a pad (position 0) and locations immediately adjacent to other stockpiles on a pad are considered (respecting the minimum distance that must be maintained between adjacent stockpiles). Two tie-breaking rules are employed: (1) if the earliest time a coal movement can take place is the same for two or more locations, then the location at the southernmost pad is given preference, and (2) if the earliest time a coal movement can take place is the same for two or more locations on the same pad, then the location with the lowest position is given preference. In Fig. 4, we give an example schedule of stockpiles on a pad. There are eight stockpiles (SP 1–SP 8) located on this pad whose coal movements and reclaiming have already been scheduled. Time is presented on the x-axis, so that the width of a rectangle represents the time period during which a stockpile is on the pad, i.e., the time at which the first coal movement for the stockpile arrives to the time at which the stockpile is completely reclaimed. The length of the pad is presented on the y-axis, so that the height of a rectangle represents the space occupied by the stockpile on the pad. Consider, for example, stockpile SP 1. The first coal movement for the stockpile arrives at time 0, it is completely reclaimed at time 86, and it occupies positions 0 through 310 on the pad. The stockpile currently being processed, SP 9, will be positioned immediately adjacent to SP 4, and the first coal movement for SP 9 can take place at time 48.
- Once the location for stockpile s has been decided, schedule its coal movements. For each load point $l \in L(s)$, in non-increasing order of the amount of coal sourced from load point l for stockpile s, schedule as many coal movements as possible from load point l for stockpile s as early as possible until all coal movements for s have been scheduled. In Fig. 5, we give an example schedule of coal movements for the stockpiles given in Fig. 4. For each stockpile, we give in parentheses the number of coal movements scheduled on a particular day for

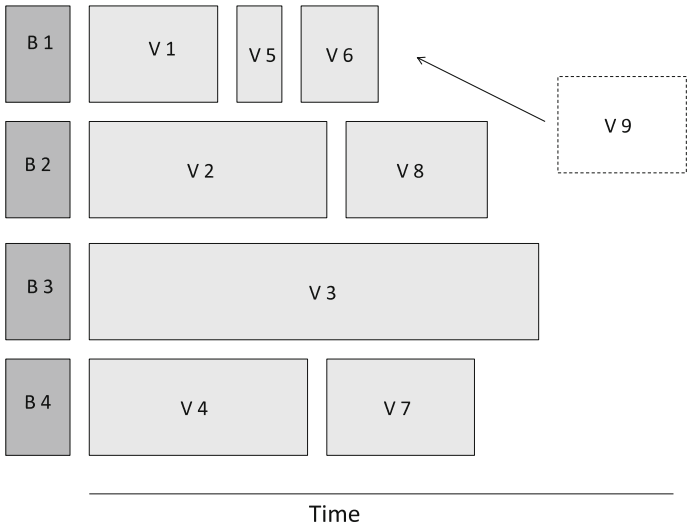

Fig. 3 We schedule ship V9 so that it can arrive as early as possible

the stockpile. We point out that not all coal movements shown for a stockpile on a particular day necessarily come from the same load point (or deliver the same amount of coal). When scheduling coal movements, daily load point and stacker stream capacities have to be respected. For stockpile SP 9, we see that one coal movement is scheduled on the third day, five coal movements are scheduled for the fourth day, one coal movement is scheduled for the fifth day, and a final coal movement is scheduled for the sixth day. As a result, stockpile SP 9 is ready to be reclaimed at the start of the seventh day.

- For each stockpile $s \in S(v)$, in the prescribed reclaiming order, assign a reclaimer $r \in R(p_s)$ that can start the reclaiming process as early as possible and schedule the reclaiming.
- Schedule the departure of ship v. If ship v does not have to depart on the high tide, set its departure time to the time that the loading of the ship completes plus the time required for paperwork processing. If ship v has to depart on the high time, set its departure time to the earliest high tide time after the loading of the ship (and paperwork processing) completes at which no other ship is departing.

In Algorithm 1, we give the pseudo-code for GCA-ETA. It calls three procedures: *Locate*, which locates a given stockpile in the stockyard and returns the earliest time a coal movement can take place for the stockpile at its location, *Schedule_Coal_Movements*, which schedules the coal movements for a given stockpile, and *Set_Reclaim_Schedule*, which schedules the reclaiming of the stockpiles. In the "Appendix", we give the pseudo-code for these procedures (Procedures 1, 2, 3). Note that the pseudo-code is meant to provide insight into the logic of the algorithms and procedures. It is not necessarily representative of the

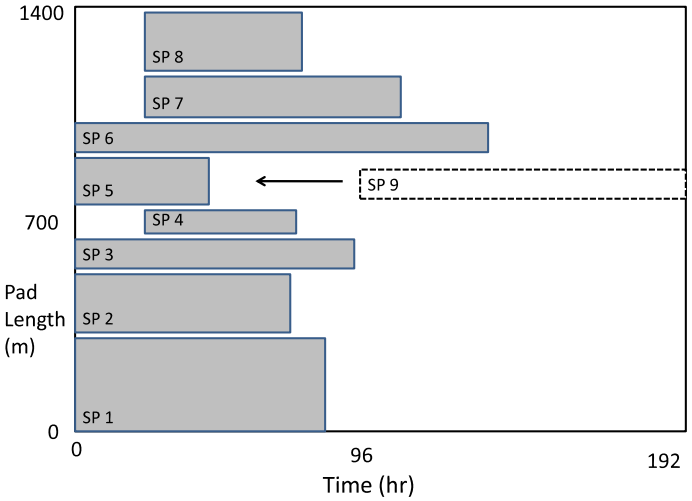

Fig. 4 We locate SP 9 so that it can start being built as early as possible

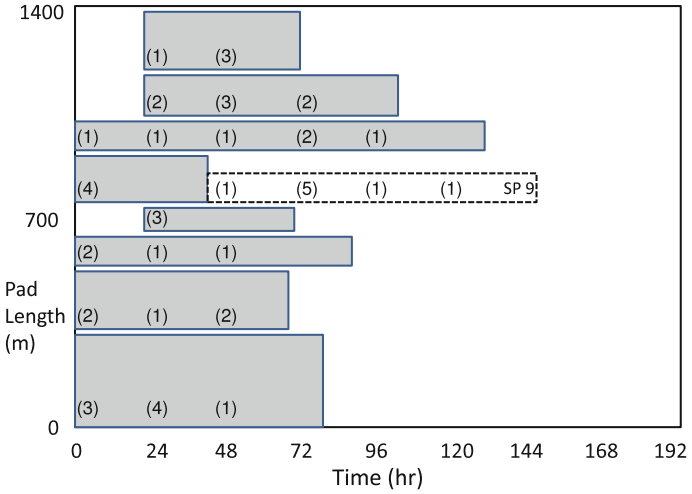

Fig. 5 We schedule the coal movements for SP 9 as early as possible in a greedy manner

actual implementation of the algorithms and procedures and leaves out many details and complexities. For example, in the procedure *Get_Reclaim_Time* (Procedure 4), which is called from *Set_Reclaim_Schedule* to determine the earliest time the reclaiming of a stockpile can start for a given reclaimer, it is necessary to ensure that no reclaimer clashes occur and that at no point in time the number of active reclaimers exceeds the maximum allowed, which is highly involved.

Algorithm 1 GCA-ETA

```
1: order ships in V by eta_v
2: for v ∈ V do
3:    best_berth = −1
4:    min_time_arr = ∞
5:    for b ∈ B do
6:       time_arr = first available time at b after eta_v
7:       if time_arr < min_time_arr then
8:          best_berth = b
9:          min_time_arr = time_arr
10:      end if
11:   end for
12:   b_v = best_berth
13:   t_v^arr = min_time_arr
14:   order stockpiles in S(v) by reclaiming sequence
15:   for s ∈ S(v) do
16:      early_time = Locate(s)
17:      Schedule_Coal_Movements(s, early_time)
18:   end for
19:   Set_Reclaim_Schedule(v)
20:   if v ∈ V^L then
21:      for h ∈ T^L such that h ≥ t_v^end + buf_{b_v}^ltd do
22:         if t_{v'}^dep ≠ h, ∀ v' ∈ V^L then
23:            t_v^dep = h
24:            break
25:         end if
26:      end for
27:   else
28:      t_v^dep = t_v^end + buf_v^ltd
29:   end if
30: end for
```

An obvious weakness of GCA-ETA is that the order in which ships are scheduled is independent of the state of the system. A better solution may be obtained by taking into account relevant available capacities when determining the next ship to be scheduled. One way to do this is to consider not only *ETA*, but also the time that loading can begin, the "time to start loading" (*TSL*), of the ships during the scheduling process. We elaborate on this idea in the next section.

A TSL-based greedy construction algorithm

Instead of selecting the ship with the smallest *ETA* to be scheduled next, we select the ship with the smallest weighted sum $\lambda\ eta_v + t_v^{beg}$ to be scheduled next, where λ is a preset parameter and t_v^{beg} is the current *TSL* of ship v. Given the current state of the system, we calculate t_v^{beg} for an as of yet unscheduled ship v by temporarily scheduling v using the steps in GCA-ETA (lines 3–19 in Algorithm 1). That is, we assign ship v to a berth, locate its stockpiles in the stockyard, schedule the coal movements for the stockpiles, and schedule their reclaiming, which gives us t_v^{beg} (the current *TSL* of v).

Unlike a ship's *ETA*, a ship's *TSL* depends on the current state of the system. As a result, the *TSL* of an unscheduled ship changes over time and needs to be recalculated each time another ship is (permanently) scheduled. This makes selecting the ship to be scheduled next using a weighted sum of *ETA* and *TSL* more computationally expensive

than selecting the ship to be scheduled next just using *ETA*. However, our computational experiments demonstrate that higher-quality solutions are obtained (i.e., the average delay is smaller) using a weighted sum of *ETA* and *TSL*. This is likely because *TSL* is a better indicator of a ship's departure time than is *ETA*, so by taking into account the *TSL*s of ships, we are effectively scheduling earlier departures for the ships. It is of course possible to select the ship to be scheduled next using the departure time of a ship directly, but we found that this biases the selection too strongly towards smaller ships—those with relatively few stockpiles to reclaim, and those that do not need to wait for a high tide to depart—unfairly causing long delays for larger ships.

We denote the *TSL*-based greedy construction algorithm by GCA-TSL. We point out that when $\lambda = \infty$, GCA-TSL is equivalent to GCA-ETA, and when $\lambda = 0$, *ETA* is not explicitly considered at all. Thus, if λ is large, there is little difference between the solutions generated by GCA-TSL and GCA-ETA. If λ is small, GCA-TSL will not give much weight to how long a ship has been "waiting" to be scheduled (the difference between the current time and a ship's *ETA*). This leads to solutions with low average delays, but unacceptably large maximum delays. Experimentation revealed that $\lambda = 0.3$ provides a proper balance.

In GCA-TSL, we use the procedures of GCA-ETA to compute the *TSL* for an unscheduled ship in *V*. We then schedule a ship v with smallest $t_v^{beg} + \lambda\, eta_v$. We repeat this process until all ships are scheduled.

When calculating the *TSL*s of ships, we do so in non-decreasing order of *ETA*s. This way, as soon as we encounter a ship v with an *ETA* such that $(1+\lambda)\, eta_v \geq t_{v^*}^{beg} + \lambda\, eta_{v^*}$, where v^* is the ship with the smallest value $t_{v^*}^{beg} + \lambda\, eta_{v^*}$ so far, then we can stop and (permanently) schedule v^*. This follows from the fact that $(1+\lambda)\, eta_v$ is a lower bound on $t_v^{beg} + \lambda\, eta_v$ because $t_v^{beg} \geq eta_v$, and from observing that since we process ships in non-decreasing order of *ETA*s, we know that we will not find a ship v with a smaller value $t_v^{beg} + \lambda\, eta_v$ than ship v^*. The pseudo-code for GCA-TSL is given in Algorithm 2.

Algorithm 2 GCA-TSL Basic

```
1: Ṽ = V {initialize set of unscheduled ships}
2: order ships in Ṽ by eta_v
3: while Ṽ ≠ ∅ do
4:    min_sum = ∞
5:    best_ship = −1
6:    for v ∈ Ṽ do
7:       if (1 + λ) eta_v ≥ min_sum then
8:          break
9:       end if
10:      partially schedule v using greedy procedure {lines 3 – 19 of Algorithm 1}
11:      if t_v^beg + λ eta_v < min_sum then
12:         best_ship = v
13:         min_sum = t_v^beg + λ eta_v
14:      end if
15:      undo partial schedule for v
16:   end for
17:   schedule best_ship using greedy procedure {lines 3 – 29 of Algorithm 1}
18:   remove best_ship from Ṽ
19: end while
```

We note that although Algorithm 2 is straightforward and intuitive, it is not very efficient. In each major iteration, we calculate the *TSL*s of many unscheduled ships. We can improve the efficiency by observing that the *TSL* of a ship calculated in one iteration provides a lower bound on the *TSL* calculated in a subsequent iteration. The reason for this is that a ship's *TSL* cannot decrease when other ships are scheduled as that only results in a reduction of resource availabilities. This allows us to significantly decrease the number of *TSL*s calculated during an execution, and thus significantly decreases the run time.

We modified the *TSL*-based greedy construction algorithm by introducing a priority queue PQ to maintain a lower bound lb_v on $t_v^{beg} + \lambda\, eta_v$ for each ship v (initially, we set $lb_v = (1 + \lambda)\, eta_v$). We let v^{min} be a ship with the smallest true value of $t_v^{beg} + \lambda\, eta_v$ (true value as opposed to a lower bound). We always evaluate a ship v with smallest lower bound lb_v. If lb_v is greater than or equal to the current best true value, then we know that we cannot find a ship with a smaller true value, so we schedule v^{min}. If lb_v is less than the current best true value, then we calculate the true value for ship v. If the true value for ship v is less than the current best true value, then we update the current best true value. If the true value for ship v is greater than or equal than the current best true value, then we update the lower bound for v (the true value becomes the new lower bound) and reinsert ship v in the priority queue. The pseudo-code for this efficient implementation is given in Algorithm 3.

Algorithm 3 GCA-TSL

```
1: PQ = ∅ {initialize the priority queue of ships}
2: for v ∈ V do
3:    lb_v = (1 + λ) eta_v {set initial lower bound}
4:    add(PQ, v)
5: end for
6: min_sum = ∞
7: best_ship = −1
8: while PQ ≠ ∅ do
9:    v = remove(PQ)
10:   if lb_v ≥ min_sum then
11:      schedule best_ship using greedy procedure {lines 3 – 29 of Algorithm 1}
12:      min_sum = ∞
13:      best_ship = −1
14:   else
15:      partially schedule v using greedy procedure {lines 3 – 19 of Algorithm 1}
16:      lb_v = t_v^beg + λ eta_v {set lower bound to true value}
17:      undo partial schedule for v
18:      if lb_v < min_sum then
19:         add(PQ, best_ship)
20:         min_sum = lb_v
21:         best_ship = v
22:      else
23:         add(PQ, v) {reinsert with updated lower bound}
24:      end if
25:   end if
26: end while
```

The enhancements discussed in this section have focused on the order in which ships are scheduled, which is one of the components of the greedy construction

algorithm that is likely to have a substantial impact on the quality of the solution. Another component that is likely to have a substantial impact on the quality of the solution is where cargoes (stockpiles) of a ship are located in the stockyard and when the coal movements for the assembly of these cargoes are scheduled. In the next section, we explore the use of enumeration and integer programming to make these decisions as opposed to using greedy rules.

An integer programming-based construction algorithm

Since integer programming (IP) models consider decisions simultaneously, as opposed to greedy algorithms that consider decisions individually and sequentially, integer programming models are usually able to produce better solutions. We exploit this advantage in scheduling the coal movements associated with the assembly of the cargoes of a ship.

Assume that a set of candidate locations is given for the stockpiles $S(v)$ of ship v, where a candidate location for a stockpile s consists of a pad p on which to assemble s and a time-space pair (t, h) indicating the earliest time t that a coal movement for s can take place and the position h on the pad p where s will be assembled. We use the following notation to describe a set of candidate locations CL_v for ship v : $CL_v = \{(s_1, p_1, (t_1, h_1)), \ldots, (s_{n_v}, p_{n_v}, (t_{n_v}, h_{n_v}))\}$, where $s_1,\ldots,s_{n_v}$ are the stockpiles in $S(v)$, p_i is a pad in P, t_i is a time in [0, T], and h_i is a position in $[0, len_{p_i}]$ for $i = 1,\ldots,n_v$.

An example of a set of candidate locations, in this case on a single pad p, is shown in Fig. 6. Stockpile SP1 has been previously scheduled. Stockpiles SP2, SP3, and SP4 are stockpiles from the same ship. Their time–space pairs on pad p are A (96, 0), B (24, 310), and C (24, 670), respectively. The candidate location set for the ship is thus {(*SP*2, *p*, 96, 0), (*SP*3, *p*, 24, 310), (*SP*4, *p*, 24, 670)}.

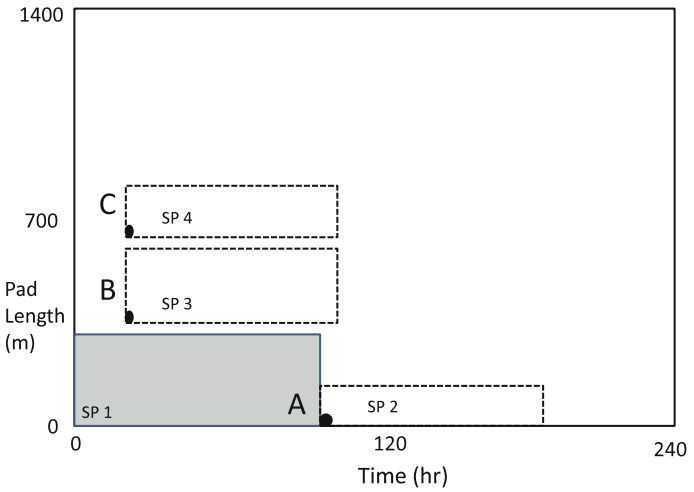

Fig. 6 An illustration of candidate locations on a pad

For each stockpile $s \in S(v)$, let $\tilde{T}(s)$ be the set of days on which coal movements for s can feasibly take place given the location of s. The first day in $\tilde{T}(s)$ is the first day on or after the time component of the time–space pair of the location of s. The last day of $\tilde{T}(s)$ is determined by finding the last day a coal movement for s can take place without conflicting with a previously scheduled stockpile. If there will never be a conflict with a previously scheduled stockpile, then we set the last day of $\tilde{T}(s)$ to be four weeks after the first day in $\tilde{T}(s)$. Let $\tilde{T} = \bigcup_{s \in S(v)} \tilde{T}(s)$ and let $S(t) = \{s \in S(v) : t \in \tilde{T}(s)\}$ for $t \in \tilde{T}$. Furthermore, let res_{lt}^{mov} be the remaining number of coal movements that can be accommodated by load point $l \in L$ on day $t \in \tilde{T}$, i.e., the maximum number of coal movements that can be accommodated by load point l on day t minus the number of coal movements from l on day t that have already been scheduled. Similarly, let res_{lt}^{ton} be the remaining tonnage of coal that can be sourced from load point $l \in L$ on day $t \in \tilde{T}$, and let res_{kt}^{dur} be the remaining number of working hours of stacker stream $k \in K$ on day $t \in \tilde{T}$. The IP model is given below.

Decision variables

x_{slt} = the number of coal movements for stockpiles $s \in S(v)$ from load point $l \in L(s)$ on day $t \in \tilde{T}(s)$

$$y_t = \begin{cases} 1 & \text{if a coal movement occurs on or after day } t \in \tilde{T} \\ 0 & \text{otherwise} \end{cases}$$

Objective

Minimize the last day that a coal movement takes place for any of the stockpiles in $S(v)$

$$\text{Minimize} \sum_{t \in \tilde{T}} t(y_t - y_{t+1})$$

Constraints

- All coal movements for a stockpile from a load point are carried out:

$$\sum_{t \in \tilde{T}(s)} x_{slt} = num_{sl}^{mov} \quad \forall s \in S(v), \forall l \in L(s)$$

- If a coal movement is carried out it must be properly accounted for:

$$\sum_{s \in S(t)} \sum_{l \in L(s)} \sum_{t' \geq t} x_{slt'} \leq num_{v}^{mov} y_t \quad \forall t \in \tilde{T}$$

- The number of coal movements from a load point does not exceed its capacity:

$$\sum_{s \in S(l) \cap S(t)} x_{slt} \leq res_{lt}^{mov} \quad \forall l \in L, \forall t \in \tilde{T}$$

- The tonnage sourced from a load point does not exceed its capacity:

$$\sum_{s \in S(l) \cap S(t)} lod_{sl}^{ton} x_{slt} \leq res_{lt}^{ton} \quad \forall l \in L, \forall t \in \tilde{T}$$

- The available working hours of a stacker stream are not exceeded:

$$\sum_{s \in S(k) \cap S(t)} \sum_{l \in L(s)} dur_{slk}^{stk} x_{slt} \leq res_{kt}^{dur} \quad \forall k \in K, \forall t \in \tilde{T}$$

We point out that the IP does not necessarily have a feasible solution. It might be the case that not all coal movements can be scheduled without exceeding capacities given the current candidate locations of the stockpiles of the ship. If the IP returns an infeasible solution, we know that we need to consider a different candidate location set.

To be able to embed the IP model into a greedy construction algorithm, we need a mechanism to generate candidate location sets for a ship v. Let $\mathcal{CL}_v$ denote a set of candidate location sets for ship v. For each $CL_v \in \mathcal{CL}_v$, we can then temporarily assign the stockpiles in $S(v)$ to the locations in CL_v, solve the IP to generate a schedule of coal movements for these stockpiles, and schedule the reclaiming of the stockpiles (this last step is done only to ensure that a feasible reclaiming schedule for v exists). From these schedules, we select the best one, i.e., the one in which ship v starts loading the earliest, and then schedule the departure of v as before.

The main challenge in this is the generation of a master set of candidate location sets. We want to generate a set of candidate location sets that is guaranteed to contain an optimal candidate location set (a candidate location set that leads to an earliest possible *TSL*), but we want to keep the set of candidate location sets as small as possible so that the algorithm remains computationally tractable. With this in mind, we generate time–space pairs for a stockpile s on a pad p by only considering times immediately after previously scheduled stockpiles on p have finished reclaiming, and by only considering positions on p directly adjacent to previously scheduled stockpiles. This avoids leaving unnecessary gaps, in time and in space, between stockpiles. We only accept a time–space pair (t, h) as a candidate for s on p, if t cannot be "shifted" to an earlier time and h cannot be shifted to a lower position on p. In addition, we can immediately rule out certain time–space pairs as candidates for stockpile s on pad p based on feasibility. For example, we can rule out as candidates those time–space pairs with a time coordinate t that is earlier than the nomination time of ship v and those with a position h that results in a physical conflict with a stockpile already scheduled.

In Fig. 7, we give a depiction in time and space of stockpiles on a pad. There are eight stockpiles (SP 1–SP 8) located on the pad. SP 9 is the stockpile for which we want to generate candidate locations. There are three candidate locations for SP 9 on the pad: location A, location B, and location C. Location X is not a candidate,

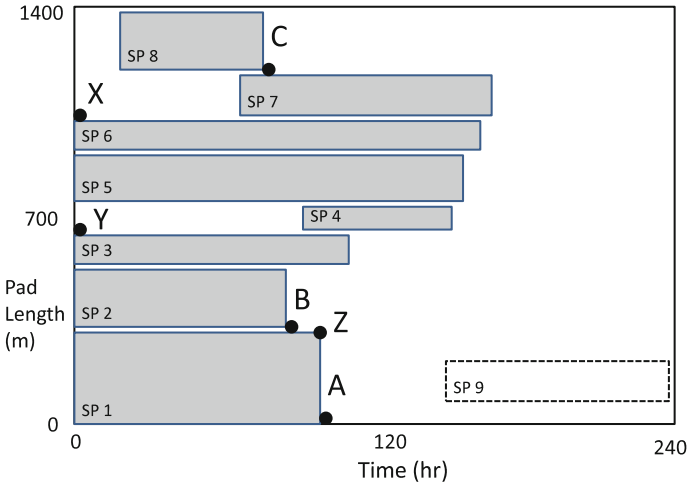

Fig. 7 An illustration of candidate locations on a pad

because locating SP 9 at X would cause a conflict with SP 7. Although we do not know a priori how long it will take to build and reclaim a stockpile (the width of the rectangle), it is easy to quickly determine a lower bound; more on this below. Based on this lower bound, location X is dismissed as a candidate for SP 9, as there would be a conflict with SP 7. Location Y is not a candidate, because locating SP 9 at Y would cause a conflict with SP 5. SP 9 takes up too much space on the pad (the height of the rectangle is too large for location Y). Location Z is not a candidate, because it can be shifted either left (to location B) or down (to location A). We point out that location Z is fundamentally different from location X and location Y. Location X and location Y are not considered as candidates because it is infeasible to locate SP 9 at these points. In contrast, it *would* be feasible to locate SP 9 at location Z. Location Z is not considered as a candidate because locating SP 9 at Z would leave an unnecessary gap in the schedule both in time and in space.

To quickly determine a lower bound on the time it will take to build stockpile s at time-space pair (t, h) on pad p, we calculate how soon after time t coal movements for stockpile s can take place from each load point in $L(s)$. The latest time that a coal movement is completed from one of the load points in $L(s)$ gives a lower bound on the time it will take to build stockpile s. When calculating the number of coal movements that can take place for s from a load point l on a given day, we ignore coal movements for s from other load points $l' \in L(s)$, so the lower bound does not generally provide a feasible schedule of coal movements for stockpile s. For example, suppose that the stacker stream servicing pad p has three available work hours on the first day at which we can start building stockpile s. Furthermore, suppose stockpile s requires a single coal movement from two load points, l and l', and each of these coal movements will result in two hours of stacking. In calculating the lower bound on the build time of stockpile s, we assume that the coal movements from both l and l' can take place on the first day, even though this is

actually infeasible as it exceeds the capacity of the stacker stream. By adding the time it takes to reclaim stockpile s to the lower bound on the time it takes to complete the coal movements, we get a lower bound on the total time stockpile s will be on the pad. This lower bound is used to rule out certain locations as candidates (for example, in Fig. 7, location X is not a candidate for SP 9). In the "Appendix", we give pseudo-code for generating the lower bound (Procedure 5), and we give pseudo-code for generating the candidate locations for stockpile s on pad p (Procedure 6).

When a ship has more than one cargo, then generating candidate location sets becomes more complicated, because the state of a pad changes when we locate a stockpile on it. For example, suppose a ship has two stockpiles s_1 and s_2. Furthermore, suppose we have generated candidate locations for stockpile s_1 and assigned stockpile s_1 to a time–space pair on pad p. The set of time–space pairs on pad p for stockpile s_2 has now likely changed due to the presence of stockpile s_1. To account for this dependency, we generate candidate locations for stockpiles sequentially in *every* possible stockpile order. With two stockpiles, this means we generate candidate locations assuming stockpile s_1 is located first followed by stockpile s_2*and* we generate candidate locations assuming that stockpile s_2 is located first followed by stockpile s_1. With three stockpiles, we would consider all six possible orderings of stockpiles (we recall that, in practice, most ships have only one or two cargoes). A recursive procedure is used for generating the master set of candidate location sets for a given ship v and a given order φ of stockpiles in $S(v)$. The procedure has a parameter k to indicate that the k^{th} stockpile in the order φ has to be processed. Processing the kth stockpile involves the following steps. For each pad p, candidate time–space pairs are generated (Procedure 6). For each time–space pair (t, h), stockpile s is temporarily scheduled at time t and position h. If s is not the last stockpile in the order, then the procedure is called recursively with argument $k + 1$ (indicating that the $k + 1$th stockpile needs to be processed), otherwise all stockpiles in $S(v)$ have been scheduled and a candidate location set has been determined and can be added to the master set of candidate locations sets for v. At the end of the procedure, the master set of candidate location sets is returned. In the "Appendix", we give pseudo-code for this procedure (Procedure 7).

We now put the pieces together to describe our IP-based scheduling procedure for a given ship v. We start by generating a master set of candidate location sets $\mathcal{CL}_v$ for ship v. For each candidate location set $CL_v \in \mathcal{CL}_v$, we try to generated a feasible schedule of coal movements of the stockpiles of v by solving the IP. We then try to reclaim the stockpiles greedily. If the coal movements can be feasibly scheduled and the stockpiles can be feasibly reclaimed and the resulting *TSL* is earlier than the *TSL* of the best schedule found so far, then we update the best schedule. We end by implementing the best schedule found throughout the process.

Again, we use easily computable lower bounds to improve efficiency. We cannot start loading ship v until all of its stockpiles have been completely built. The earliest time a stockpile $s \in S(v)$ can possibly finish being built is one day after its first coal movement takes place. The first coal movement of stockpile s cannot take place prior to time t, where t is the time coordinate of the time–space pair of the candidate location of stockpile s. Therefore, it is easy to calculate a lower bound on the time

the loading of ship v can start for a particular candidate location set. We explore candidate location sets in order of non-decreasing lower bounds. When a candidate location set has a lower bound larger than the *TSL* of the best schedule found so far, then we can stop because we know we have identified the best schedule. In the "Appendix", we give pseudo-code for the IP-based scheduling procedure for a ship (Procedure 8).

Using the IP-based scheduling procedure for a ship, we derive the IP-based greedy construction algorithm (GCA-IP). GCA-IP follows the steps of GCA-TSL (Algorithm 3), except that when the stockpiles of a ship are located and the coal movements for the stockpiles are scheduled, we use the IP-based procedure instead of the greedy procedure. We limit the use of the IP-based procedure to ships with no more than two stockpiles. The reason for this is that in practice this captures the vast majority of the ships and the generation of candidate location sets for ships with more than two stockpiles can be computationally expensive. In Algorithm 4, we give pseudo-code for CGA-IP. CGA-IP takes the parameter η, and the IP-based ship scheduling procedure (Procedure 8) is applied to ships with η or fewer stockpiles. As stated above, in practice, we let $\eta = 2$.

A computational study

In this section, we report on a computational study conducted to evaluate the performance of the three greedy construction algorithms, to assess the benefits of the efficiency improvements incorporated in the algorithms, and to demonstrate the value of the algorithms in identifying capacity expansion opportunities along the coal chain.

The computational study was conducted using 14 SPP instances. The 14 instances are derived from two related, but different infrastructures, where an infrastructure defines the parameters related to the physical aspects of the system, e.g., the number of berths, the number of pads in the stockyard, the number of load points, the number of reclaimers, and the rate at which reclaimers operate. For each infrastructure, there are seven different ship arrival streams, where a ship arrival stream defines the parameters related to the ships and its cargoes, e.g., a ship's *ETA*, the number of cargoes, the cargo tonnages, and the cargo recipes.

The two infrastructures are based on the characteristics of the HVCC in the years 2006 and 2008. The seven different ship arrival streams for an infrastructure are derived from the historic ship arrivals for the first 6 months of the corresponding year. To obtain seven different ship arrival streams, we compressed the *ETAs* of the ships, so that the ships nominate more frequently. When the ships nominate more frequently, more strain is put on the system, which allows us to analyze how the different algorithms respond under different levels of stress.

To compress *ETAs*, we order the ships, v_1, v_2,…, v_n, by *ETA* from the earliest to the latest. We let $\Delta_i = eta_{v_i} - eta_{v_{i-1}}$ be the expected interarrival time between ships v_i and $v_{i\text{-}1}$. We then let the new *ETA* of ship v_i be given by $eta_{v_i}^{new} = eta_{v_{i-1}}^{new} + \alpha_i \Delta_i$, where $eta_{v_1}^{new} = eta_{v_1}$. We periodically change the value of α_i to create periods of

Algorithm 4 GCA-IP(η)

1: $PQ = \emptyset$ {initialize the priority queue of ships}
2: **for** $v \in V$ **do**
3: $lb_v = (1 + \lambda)\ eta_v$ {set initial lower bound}
4: add(PQ, v)
5: **end for**
6: $min\_sum = \infty$
7: $best\_ship = -1$
8: **while** $PQ \neq \emptyset$ **do**
9: v = remove(PQ)
10: **if** $lb_v \geq min\_sum$ **then**
11: assign berth to *best_ship* using greedy procedure {lines 3 – 13 of Algorithm 1}
12: **if** $|S(best\_ship)| \leq \eta$ **then**
13: IP-based_Stockpile_Scheduler(*best_ship*)
14: **else**
15: partially schedule *best_ship* using greedy procedure {lines 14 – 19 of Algorithm 1}
16: **end if**
17: schedule departure of *best_ship* using greedy procedure {lines 20 – 29 of Algorithm 1}
18: $min\_sum = \infty$
19: $best\_ship = -1$
20: **else**
21: partially schedule v using greedy procedure {lines 3 – 19 of Algorithm 1}
22: $lb_v = t_v^{beg} + \lambda\ eta_v$ {set lower bound to true value}
23: undo partial schedule for v
24: **if** $lb_v < min\_sum$ **then**
25: add(PQ, *best_ship*)
26: $min\_sum = lb_v$
27: $best\_ship = v$
28: **else**
29: add(PQ, v)
30: **end if**
31: **end if**
32: **end while**

compression and periods of non-compression, i.e., for the first q ships, we let $\alpha_i = \alpha$ ($0 < \alpha \leq 1$), for the next q ships we let $\alpha_i = 1$, for the next q ships we again let $\alpha_i = \alpha$, and so on. We used periodic compression to create periods of high stress and normal stress on the system. In practice, high stress periods are caused by things that we do not explicitly model in the SPP, such as railroad track and machine maintenance, and delays caused by inclement weather. For each infrastructure, we used seven different compression scenarios in our experiments: $\alpha = 1$, $q = n$ (no compression), and each combination of $\alpha = 0.5$, 0.75 and $q = \lceil \frac{n}{2} \rceil, \lceil \frac{n}{4} \rceil, \lceil \frac{n}{6} \rceil$.

Also, in each scenario, we added a fixed number of days τ to the *ETA*s of all the ships. The reason for this is that since there cannot be any coal movements prior to the first day of the planning horizon ($t = 0$), ships arriving very early in the planning horizon can have artificially long delays, as there is little or no time prior to their arrivals to build their stockpiles. By adding a constant τ to all *ETAs*, we ensure that there will be a period of at least τ days prior to a ship's arrival during which coal movements for the stockpiles of the ship can take place. Note that with $\tau = 0$, we essentially assume that the stockyard is completely full until time $t = 0$. On the other hand, with $\tau \geq 10$, we essentially assume that the stockyard is

Table 1 Instance characteristics

	2006	2008
Number of ships	339	352
Number of one-stockpile ships	272	266
Number of two-stockpile ships	24	37
Number of three-stockpile ships	20	24
Number of four-stockpile ships	16	19
Number of five-stockpile ships	6	5
Number of six-stockpile ships	1	1
Number of stockpiles	480	519
Average number of coal movements per stockpile	10.39	11.61
Average stack duration per coal movement (h)	1.26	0.94
Average reclaim duration per stockpile (h)	16.97	9.73

completely empty for the first arriving ship (a ship nominates at least 10 days prior to its *ETA*). In our computational experiments, we let $\tau = 3$ so that early-arriving ships will see a stockyard that is neither artificially full nor artificially empty.

In Tables 1, 2, we present characteristics of the 14 instances. In Table 1, we present information regarding the ships and stockpiles. In Table 2, we present information regarding the interarrival time distributions for the ship streams. In the first column of Table 2, we give time intervals (in hours). In columns two through eight, we give the number of interarrival times in each interval for the given ship stream. For example, for the ship stream of 2006 with no compression there are 109 interarrival times between 0 and 5 h.

We see that, as mentioned earlier, the vast majority of ships have only a single cargo and very few have more than four cargoes. The stacking and reclaiming times are averages over all stockpiles and all reclaimers. These values are meant primarily to give an indication and feel for the time involved in stacking a train load and reclaiming a stockpile. However, it is important to observe that the average time it takes to stack a train load and, especially, the average time to reclaim a stockpile are quite different for the two settings. This is because the effective rates at which the stockyard equipment operates, i.e., the stackers and the reclaimers, has increased from 2006 to 2008 as a result of mechanical improvements.

Finally, we note that all experiments were run on a Dell PowerEdge 2950 with dual quad core 3.16 GHz Intel Xeon X5460 processors and 64 GB of RAM. The IPs in GCA-IP were solved using CPLEX 12.1.

Algorithm performance

In Table 3, we give basic statistics for the solutions generated by GCA-ETA, GCA-TSL, and GCA-IP for the 14 instances, namely the mean, median, and maximum delays (in hours) of the ships. Figures 8, 9 provide more detail. Each graph shows for each of the greedy construction algorithms the delay of each of the ships where the ships are given in order of non-decreasing delays; top—GCA-ETA, middle—

Table 2 Distributions of the interarrival times for the ship arrival streams

2006 infrastructure							
Time interval	No. compression	$\alpha = 0.5$			$\alpha = 0.75$		
		$q = 57$	$q = 85$	$q = 170$	$q = 57$	$q = 85$	$q = 170$
[0, 5)	109	146	152	139	121	118	118
[5, 10)	70	79	68	96	73	77	84
[10, 15)	47	42	48	40	57	51	52
[15, 20)	41	24	28	16	28	39	27
[20, 25)	27	20	16	18	24	22	22
[25, 30)	16	12	11	11	15	10	10
[30, 35)	8	4	5	5	7	6	8
[35, 40)	6	2	3	5	3	5	7
[40, 45)	2	1	3	1	1	4	1
[45, 50)	4	3	2	3	3	3	3
[50, 55)	6	4	2	3	4	2	3
$[55, \infty)$	2	1	0	0	1	1	1
2008 Infrastructure							
Time interval	No. compression	$\alpha = 0.5$			$\alpha = 0.75$		
		$q = 59$	$q = 88$	$q = 176$	$q = 59$	$q = 88$	$q = 176$
[0, 5)	109	156	161	158	123	130	129
[5, 10)	87	82	74	81	85	79	83
[10, 15)	49	52	50	47	58	53	52
[15, 20)	35	18	25	20	31	33	28
[20, 25)	23	12	13	15	20	23	27
[25, 30)	22	12	12	10	12	14	10
[30, 35)	9	9	6	8	8	6	6
[35, 40)	3	3	1	3	3	2	4
[40, 45)	5	3	3	5	3	3	4
[45, 50)	1	1	1	1	2	1	2
[50, 55)	3	2	2	2	4	4	4
$[55, \infty)$	5	1	3	1	2	3	2

GCA-TSL, bottom—GCA-IP. For example, in Fig. 8a, there is a top line data point at (50, 333.09), a middle line data point at (50, 135.03), and a bottom line data point at (50, 24.53). This means that the ship with the 50th largest delay in the solution generated by GCA-ETA was delayed 333.09 h, the ship with the 50th largest delay in the solution generated by GCA-TSL was delayed 135.03 h, and the ship with the 50th largest delay in the solution generated by GCA-IP was delayed 24.53 h. The results show that the solutions generated by GCA-TSL are far superior to the solutions generated by GCA-ETA, and the solutions generated by GCA-IP are far superior to the solutions generated by GCA-TSL. Over all 14 instances, on

Table 3 Mean, median, and average vessel delays (in hours) of solutions generated by three algorithms on 14 instances

α	q	GA-ETA			GCA-TSL			GCA-IP		
		Mean	Median	Max	Mean	Median	Max	Mean	Median	Max
2006 infrastructure										
1	339	219.92	233.32	397.77	80.10	77.90	259.40	9.71	2.20	161.22
0.75	57	507.00	571.03	871.86	250.98	249.01	465.88	27.48	19.17	273.12
0.75	85	550.19	559.04	921.35	305.04	286.03	582.87	18.16	11.15	198.60
0.75	170	601.04	737.06	893.31	341.30	373.53	670.93	36.20	17.12	359.85
0.5	57	832.57	920.20	1453.87	388.48	420.83	679.12	127.15	122.62	409.27
0.5	85	919.36	855.35	1554.22	506.35	461.60	895.46	152.30	125.32	348.35
0.5	170	1020.87	1269.70	1490.97	560.00	634.93	994.65	255.64	270.75	710.00
2008 infrastructure										
1	352	12.26	3.13	127.27	12.48	5.58	127.60	5.10	0.00	173.90
0.75	59	37.07	15.08	198.53	20.18	12.47	335.45	6.91	0.00	335.45
0.75	88	36.40	13.26	198.53	17.82	10.98	335.45	7.70	0.00	335.45
0.75	176	47.68	24.96	198.53	28.08	14.62	335.45	8.46	0.00	335.45
0.5	59	116.12	86.81	312.84	64.04	56.92	423.30	21.02	12.12	447.30
0.5	88	143.62	136.98	402.22	59.90	37.79	507.50	23.83	6.81	532.57
0.5	176	271.50	274.43	581.03	121.63	100.32	423.30	44.80	13.46	447.30

average, the solutions generated by GCA-TSL have a mean delay that is 45 % less than the solutions generated by GCA-ETA, a median delay that is 38 % less, and a maximum delay that is 2 % less. The solutions generated by CGA-IP have a mean delay that is 71 % less than the solutions generated by GCA-TSL, a median delay that is 88 % less, and a maximum delay that is 19.17 % less. We further observe that the delays for the 2008 instances are much smaller than the delays for the 2006 instances. This is most likely the result of the improvement in the effective operating rates of the stackers and reclaimers.

In Table 4, we give run times for the three algorithms. We see that GCA-ETA is by far the fastest algorithm, with an average run time of 35.17 s across all 14 instances. The average run times for GCA-TSL and GCA-IP were 191.44 and 247.27 s, respectively. We point out that the run time of GCA-ETA is essentially independent of the compression type, since exactly one schedule is evaluated per iteration in GCA-ETA. The run times of GCA-TSL and GCA-IP are dependent on compression type, because greater compression usually means more schedules have to be evaluated per iteration.

Efficiency engineering

To evaluate the gains in efficiency from algorithm engineering, i.e., employing lower bounds and a priority queue in the selection of the ship to be scheduled next and employing lower bounds on the *TSL* for a given candidate location set in the IP-based stockpile scheduling, we implemented a stripped-down version of GCA-IP, referred to as GCA-IP Basic, which does not have these features.

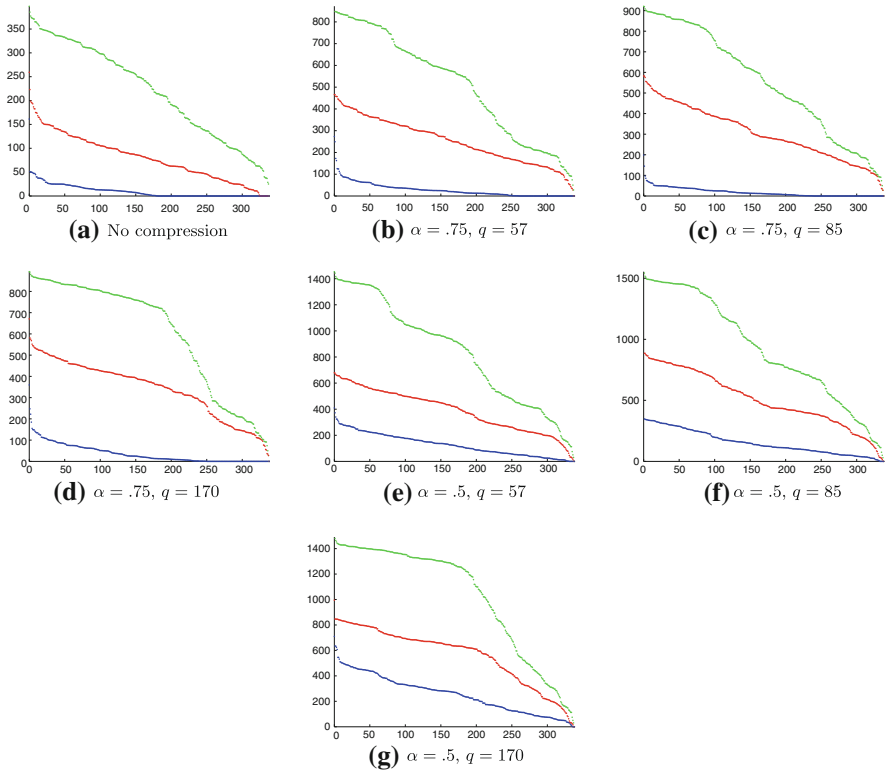

Fig. 8 Graphs of ship delays (in hours) for the solutions generated by the three algorithms for the 2006 infrastructure under seven different compression settings: GCA-ETA (*top*), GCA-TSL (*middle*), and GCA-IP (*bottom*)

We applied GCA-IP Basic to the 14 instances and for each execution, we recorded the run time, the total number of schedules evaluated (total number of ships scheduled temporarily or permanently), and the total number of IPs solved. We compared these statistics to the ones generated for GCA-IP on these instances. In Table 5, we give the results.

We see that GCA-IP is much more efficient than GCA-IP Basic. The average run time for GCA-IP (247.42 s) was 27 % less than the average run time for GCA-IP Basic (337.12 s). The average number of schedules evaluated in GCA-IP (1,269.86) was 45 % less than the average number of schedules evaluated in GCA-IP Basic (2,317.07), and the average number of IPs solved in GCA-IP (6,364.79) was 7 % less than the average number of IPs solved in GCA-IP Basic (6,815.57).

Notice that the largest gain in efficiency is due to a reduction in the number of schedules evaluated, meaning that the use of the priority queue in Algorithms 3 and 4 is very effective. Also, these gains in efficiency become more pronounced as the level of compression increases. When there is no compression, the decrease in the number of schedules evaluated from GCA-IP Basic to GCA-IP is 11 % for the 2006 infrastructure and 6 % for the 2008 infrastructure. For the compression setting that leads to the greatest average delay ($\alpha = 0.5, q = \lceil \frac{n}{2} \rceil$), the decrease in the number of schedules evaluated

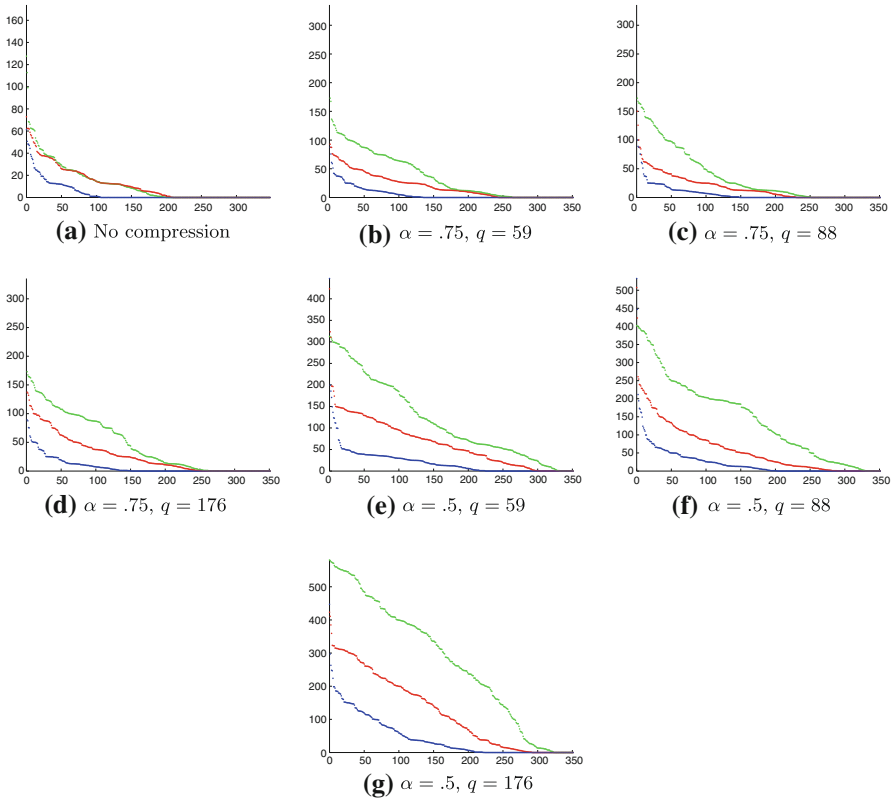

Fig. 9 Graphs of ship delays (in hours) for the solutions generated by the three algorithms for the 2008 infrastructure under seven different compression settings. GCA-ETA (*top*), GCA-TSL (*middle*), and GCA-IP (*bottom*)

from GCA-IP Basic to GCA-IP is 65 % for the 2006 infrastructure and 49 % for the 2008 infrastructure. This leads to a greater decrease in run time as compression increases. When there is no compression, the decrease in run time from GCA-IP Basic to GCA-IP is 11 % for the 2006 infrastructure and 1 % for the 2008 infrastructure. When $\alpha = 0.5$ and $q = \lceil \frac{n}{2} \rceil$, the decrease in run time is 47 % for the 2006 infrastructure and 15 % for the 2008 infrastructure. The difference in the number of schedules evaluated in 2006 and 2008 is most likely the result of the fact that the delays in 2008 are much smaller. When the delays are larger, the number of schedules that need to be evaluated increases. As the demand for coal increases and the stockyard system becomes more strained, the need for fast and effective algorithms will become more pronounced and efficiency engineering will become more important.

Identifying bottlenecks

One area of research in which the HVCCC is very interested is the identification of bottlenecks within the coal chain. In planning future modifications to the chain, it is

Table 4 Run times of three algorithms on 14 instances

α	q	Run time(s)		
		GCA-ETA	GCA-TSL	GCA-IP
2006 infrastructure				
1	339	28.98	131.21	135.99
0.75	57	29.64	242.41	201.64
0.75	85	29.75	252.68	166.84
0.75	170	29.79	260.27	168.55
0.5	57	28.99	303.87	328.65
0.5	85	29.38	321.83	332.26
0.5	170	29.21	289.32	348.21
2008 infrastructure				
1	339	39.12	99.01	214.97
0.75	59	41.37	107.56	201.63
0.75	88	40.02	104.94	253.12
0.75	176	41.70	110.72	202.04
0.5	59	41.15	165.73	274.51
0.5	88	41.11	148.64	327.07
0.5	176	42.20	142.01	308.37

Table 5 Efficiency comparison of two algorithms

α	q	GCA-IP			GCA-IP Basic		
		Run time(s)	Schedules evaluated	IPs solved	Run time(s)	Schedules evaluated	IPs solved
2006 infrastructure							
1	339	135.99	852	4,197	152.60	953	4,557
0.75	57	201.64	1,090	5,496	216.28	1,319	5,681
0.75	85	166.84	972	4,803	190.28	1,204	5,042
0.75	170	168.55	1,122	5,052	212.80	1,653	6,107
0.5	57	328.65	1,988	6,550	677.24	4,677	7,158
0.5	85	332.26	2,133	6,305	686.54	5,167	7,159
0.5	170	348.21	2,602	6,625	659.70	7,496	6,823
2008 infrastructure							
1	352	214.97	810	6,183	217.61	858	6,469
0.75	59	201.63	868	5,385	218.99	975	5,782
0.75	88	253.12	863	7,353	267.74	987	7,819
0.75	176	202.04	892	5,698	227.94	1,039	6,148
0.5	59	274.51	1,119	7,146	307.00	1,733	7,646
0.5	88	327.07	1,108	9,066	323.82	1,713	9,267
0.5	176	308.37	1,359	9,248	361.09	2,665	9,760

Table 6 Solutions generated by GCA-IP for a base case and three different increases in system capacity

Capacity increase type	Solution values			Percent decrease from base case		
	Mean	Median	Max	Mean	Median	Max
2006 infrastructure						
Base case	518.76	546.63	1,115.43	0.00	0.00	0.00
Pads	475.68	498.13	1,005.18	8.30	8.87	9.88
Load points	490.11	509.68	1,054.27	5.52	6.76	5.48
Ship loaders	446.23	435.90	1,040.50	13.98	20.26	6.72
2008 infrastructure						
Base case	133.11	112.09	1,154.73	0.00	0.00	0.00
Pads	97.09	69.68	495.30	27.06	37.83	57.11
Load points	74.42	64.49	414.13	44.09	42.47	64.14
Ship loaders	131.57	99.33	919.90	1.16	11.38	20.34

vital to know which improvements will yield the most bang for the buck. With this in mind, we considered three potential bottlenecks specific to the stockyard system—the load points, the stockyard pads, and the ship loaders. To test the sensitivity of ship delay to the capacities of these aspects of the stockyard, we ran three experiments.

In the first experiment, we increased the lengths of the two short pads (Pads C and D in Fig. 1) from 1,400 to 2,450 m, so that they were the same lengths as the full-sized pads (Pads A and B in Fig. 1). In the second experiment, we increased the load point capacities by 20 %, i.e., for each $l \in L$, we let $cap_l^{ton} = 1.2\ cap_l^{ton}$. To increase the number of coal movements that can be accommodated at the load points accordingly, we let $cap_l^{mov} = \lfloor\ cap_l^{ton} / siz^{trn}\ \rfloor$. In the third experiment, we increased the number of ship loaders from three to four, i.e., we let $num^{mxr} = 4$, so that four ships could be reclaimed (by different reclaimers) at the same time instead of only three.

Since the goal in our experiments is to identify potential bottlenecks in the stockyard system in the future, as the demand for coal increases and the system becomes more strained, in each experiment, we ran GCA-IP on very compressed instances. We let $\alpha = 0.5$ and $q = n$ and considered both the 2006 and 2008 infrastructures. Thus, the interarrival times between all ships are reduced by half. In Table 6, we give our results. We ran GCA-IP four times, one for each of the three capacity increases described above, and a base case, in which capacity was unchanged. For each run, we recorded the mean, median, and maximum delays of the solutions generated by GCA-IP.

In Table 6, we see that all three types of increase in capacity decrease overall delay, but the type that gives the greatest decrease is different for the 2006 and 2008 infrastructures. For the 2006 infrastructure, adding another ship loader gives the largest decrease of delay, while for the 2008 infrastructure, adding another ship loader gives the smallest decrease of delay. Since the parameter settings are not the

same for the 2006 and 2008 infrastructures, it makes sense that different types of capacity increases could cause different decreases in delay. For example, in Table 1, we see that the average time it takes to reclaim a stockpile is 16.97 h for the 2006 infrastructure and 9.73 for the 2008 infrastructure. This discrepancy in average reclaiming durations could explain why adding another ship loader is much more beneficial in the 2006 case than in the 2008 case. The HVCCC already has plans to extend the lengths of Pads C and D by the start of 2012. Based on the results in Table 6, they might also wish to consider exploring various ways to increase load point capacities.

Final remarks

We investigated a highly complex decision problem encountered by the planners at the HVCCC. We designed and implemented an optimization-based heuristic for its solution as a first step towards developing decision support technology for those planners. The problem has features found in many well-known optimization problems, such as 2D packing and resource scheduling, but, to the best of our knowledge, there is no other problem that encapsulates all these features into a single mine-to-port optimization problem. A computational study has demonstrated the efficacy of the optimization-based heuristic and has illustrated its potential value in capacity expansion planning exercises.

The research reported can thus be viewed as a successful proof-of-concept study. Additional work is required before the technology can become part of the core of the next generation decision support technology under development at HVCCC. There are a number of practical considerations that are not yet captured, but are relatively easy to incorporate. For example, planned equipment maintenance, which results in reduced capacity during certain time periods, is not accounted for but can easily be accommodate by refining the capacity specifications. Another desire is to be able to limit the number of coal movements from a particular region. There are two main regions in the Hunter Valley, the Ulan region and the Gunnedah region, and the planners want to be able to control the balance between the coal movements coming from the two regions. The technology also needs to be embedded into a rolling horizon framework to better assess its value for tactical/operational planning, where it may be needed most to ensure that the necessary throughput increases are realized.

We are continuing to investigate algorithm enhancements and alternative approaches. We are looking at replacing other greedy components with optimization procedures, for example, the generation of reclaiming schedules, so as to further improve the quality of the plans produced by the technology. Furthermore, we are exploring post-processing optimization models, for example, a dwell-time minimization model that reduces the total on-the-ground time for stockpiles without affecting the mean delay of ships, and a two-ship optimization model that minimizes the mean delay of two consecutive ships by re-optimizing the decisions involving these two ships while keeping the decisions for all other ships fixed.

Appendix

Procedure 1 Locate(s)

$best\_pad = -1$
$best\_pos = -1$
$early\_time = \infty$
for $p \in P$ **do**
  add "dummy" stockpile $\tilde{s}$ to pad p, with $t_{\tilde{s}}^{end} = 0$ and $h_{\tilde{s}} + len_{\tilde{s}} + buf_p^{dis} = 0$
end for
$S' =$ set of stockpiles already located in the stockyard ordered by the time they are finished being reclaimed (if two stockpiles are tied, the one with the southernmost pad takes precedent, if still tied the one with the lowest position takes precedent)
for $s' \in S'$ **do**
  $t = t_{s'}^{end}$
  if $t \geq early\_time$ **then**
    break
  end if
  $p = p_{s'}$
  $S'' =$ set of stockpiles located on pad p ordered by position
  for $s'' \in S''$ **do**
    $h = h_{s''} + len_{s''} + buf_p^{dis}$
    if $h + len_s \leq len_p$ **then**
      temporarily assign stockpile s to pad p, at position h, for the entire time horizon starting at time t
      if s does not conflict with any other stockpile on p **then**
        for $\hat{t} \in \hat{T}$ such that $\hat{t} \geq t$ **do**
          if $\hat{t} \geq early\_time$ **then**
            break
          else if at least one coal movement can feasibly be carried out for s on day $\hat{t}$ **then**
            $early\_time = \hat{t}$
            $best\_pad = p$
            $best\_pos = h$
          end if
        end for
      end if
    end if
  end for
end for
$v =$ ship of stockpile s
if $early\_time < nom_v$ **then**
  $early\_time = \min\{\hat{t} \in \hat{T} : \hat{t} \geq nom_v\}$
end if
return $early\_time$

Procedure 2 Schedule_Coal_Movements($s, early\_time$)

order the load points $L(s)$ from most to least total tons of coal for stockpile s
for $l \in L(s)$ **do**
  $res\_num = num_{sl}^{mov}$ {initialize the residual number of coal movements that must be carried out}
  for $t \in \hat{T}$ such that $t \geq early\_time$ **do**
    res_trj = number of coal movements that can be carried out for stockpile s from load point l on day t without violating the daily capacity of l
    res_ton = number of coal movements that can be carried out for s from l on t without violating the daily tonnage capacity of l
    res_dur = number of coal movements that can be carried out for s from l on t without violating the daily work duration capacity of stacker stream k_s
    $res\_min = \min(res\_num, res\_ton, res\_trj, res\_dur)$
    if $res\_min > 0$ **then**
      $n_{slt}^{mov} = \min(res\_num, res\_min)$
      $res\_num = res\_num - n_{slt}^{mov}$
      update residual capacities of load point l and stacker stream k_s on day t
      if $res\_num = 0$ **then**
        break
      end if
    end if
  end for
end for

Procedure 3 Set_Reclaim_Schedule(v)

$early\_time = \max\{\max\{t_s^{lst} : s \in S(v)\}, t_v^{arr} + buf_v^{atl}\}$ {set the earliest time we can possibly start reclaiming the first stockpile of ship v to be after all stockpiles have been built and v is at berth ready to be loaded}
for $i = 1$ to n_v **do**
  if $i > 1$ **then**
    $early\_time = t_{s_{v(i-1)}}^{end}$ {if s is not the first stockpile in the reclaiming order, update the earliest time to be after the previous stockpile finishes being reclaimed}
  end if
  $best\_rcl\_time = \infty$
  $best\_rcl = -1$
  for $r \in R(p_{s_{vi}})$ **do**
    $rcl\_time = Get\_Reclaim\_Time(s, r, early\_time)$
    if $rcl\_time < best\_rcl\_time$ **then**
      $best\_rcl\_time = rcl\_time$
      $best\_rcl = r$
    end if
  end for
  $r_{s_{vi}} = best\_rcl$
  schedule reclaiming of s_{vi}
end for

Procedure 4 *Get_Reclaim_Time*$(s, r, early\_time)$

$S'(r)$ = set of stockpiles that finish being reclaimed by r after $early\_time$
order the stockpiles in $S'(r)$ by reclaim time, i.e., let $S'(r) = \{s'_1, s'_2, ..., s'_m\}$, where r finishes reclaiming s'_i prior to reclaiming s'_{i+1}
if $t^{beg}_{s'_1} \leq early\_time$ **then**
  $early\_time = t^{end}_{s'_1} + dur^{mov}_{rh^{mid}_{s'_1} h^{mid}_s}$
  remove s'_1 from $S'(r)$, relabel accordingly {we can't start reclaiming s using r at a time immediately prior to s'_1, so we remove s'_1 from the set}
end if
initialize time interval $[a, b] = [early\_time, \infty]$ {$[a, b]$ will represent a "gap" in the reclaiming schedule of r}
for $i = 1$ to m **do**
  if $i > 1$ **then**
    $a = t^{end}_{s'_{i-1}} + dur^{mov}_{rh^{mid}_{s'_{i-1}} h^{mid}_s}$
  end if
  $b = t^{beg}_{s'_i} - dur^{mov}_{rh^{mid}_s h^{mid}_{s'_i}}$
  if $dur^{rec}_s \leq b - a$ **then**
    if we can reclaim stockpile s using reclaimer r during $[a, b]$ avoiding clashes with other reclaimers and never using more than num^{mxr} reclaimers at once **then**
      return earliest such time
    end if
  end if
end for
return earliest time after $t^{end}_{s'_m} + dur^{mov}_{rh^{mid}_{s'_m} h^{mid}_s}$ at which we can start reclaiming stockpile s using reclaimer r avoiding clashes with other reclaimers and never using more than num^{mxr} reclaimers at once {we return a time after r finishes reclaiming its last stockpile scheduled thus far}

Procedure 5 Lower_Bound_Build_And_Reclaim_Time(s, t, h)

$t^{max} = \infty$ {initialize the maximum earliest day for which all the coal movements from a load point for s can be carried out}
for $l \in L(s)$ **do**
  $res\_num = num^{mov}_{sl}$ {initialize the number of coal movements remaining from this load point}
  for $t' \in \hat{T}$ such that $t' \geq t$ **do**
    res_trj = number of coal movements that can be carried out for stockpile s from load point l on day t' without violating the daily capacity of l
    res_ton = number of coal movements that can be carried out for s from l on t' without violating the daily tonnage capacity of l
    res_dur = number of coal movements that can be carried out for s from l on t' without violating the daily work duration capacity of stacker stream k_s
    $res\_min = \min(res\_num, res\_ton, res\_trj, res\_dur)$
    if $res\_min > 0$ **then**
      $res\_num = res\_num - res\_min$
      {do *not* update residual capacities of load point l and stacker stream k_s on day t'}
      if $res\_num = 0$ **then**
        if $t' > t^{max}$ **then**
          $t^{max} = t'$
        end if
        break
      end if
    end if
  end for
end for
return $t^{max} + dur^{rec}_s$

Procedure 6 Candidate_Locations_Stockpile(s, p)

$CL_{sp} = \emptyset$ {initialize the candidate locations for stockpile s on pad p}
$X = \emptyset$ {initialize the array of time coordinates}
$Y = \emptyset$ {initialize the array of position coordinates}
add "dummy" stockpile $\tilde{s}$ to pad p, with $t_{\tilde{s}}^{end} = 0$ and $h_{\tilde{s}} + len_{\tilde{s}} + buf_p^{dis} = 0$
for $s' \in S(p)$ **do**
  if $t_{s'}^{end} \geq nom_{v_s}$ **then**
    add $t_{s'}^{end}$ to X
  end if
  if $(h_{s'} + len_{s'} + buf_p^{dis} + h_s \leq len_p)$ **then**
    add $h_{s'} + len_{s'} + buf_p^{dis}$ to Y
  end if
end for
for $t \in X$ **do**
  for $h \in Y$ **do**
    if we cannot "shift" (t, h) left or down **then**
      t^{lb} = Lower_Bound_Build_And_Reclaim_Time(s, t, h) + dur_s^{rec}
      if s does not conflict with any stockpiles on p at h from t to t^{lb} **then**
        add (t, h) to CL_{sp}
      end if
    end if
  end for
end for
return CL_{sp}

Procedure 7 Candidate_Locations_Ship(v, φ, k)

if $k = 1$ **then**
  $CL_v = \emptyset$ {initialize current location set for ship v}
  $\mathcal{CL} = \emptyset$ {initialize the master set of candidate location sets}
end if
if $k > |S(v)|$ **then**
  add CL_v to $\mathcal{CL}$
  return $\mathcal{CL}$
else
  $s = k$-th stockpile in permutation φ
  for $p \in P$ **do**
    CL_{sp} = Candidate_Locations_Stockpile(s, p)
    for $(t, h) \in CL_{sp}$ **do**
      temporarily schedule stockpile s at position h on pad p at time t
      add (s, p, t, h) to CL_v
      $\mathcal{CL} = \mathcal{CL} \bigcup$ Candidate_Locations_Ship$(v, \varphi, k + 1)$
      un-schedule s
      remove (s, p, t, h) from CL_v
    end for
  end for
end if
return $\mathcal{CL}$

Procedure 8 IP-based_Stockpile_Scheduler(v)

1: $\mathcal{CL} = \emptyset$ {initialize the master set of candidate location sets}
2: **for** Each permutation φ of stockpiles in $S(v)$ **do**
3: $\mathcal{CL} = \mathcal{CL} \bigcup$ Candidate_Locations_Ship$(v, \varphi, 1)$
4: **end for**
5: **for** $CL \in \mathcal{CL}$ **do**
6: $lb_{CL} =$ min $\{\hat{t} \in \hat{T} : \hat{t} \geq t \ \ \forall\, (s, p, t, h) \in L, \ \ \forall\, s \in S(v)\} + 1$ day
{set lower bound on the TSL of v for candidate location set CL}
7: **end for**
8: sort $\mathcal{CL}$ in non-decreasing order of lb_{CL}
9: $min\_tsl = \infty$
10: **for** $CL \in \mathcal{CL}$ **do**
11: **if** $lb_{CL} \geq min\_tsl$ **then**
12: **break**
13: **end if**
14: solve IP for ship v at candidate location set CL
15: **if** IP returns a feasible solution **then**
16: schedule coal movements for stockpiles in $S(v)$ from IP solution
17: schedule reclaiming of v using greedy procedure {line 19 of Algorithm 1}
18: $sched =$ current schedule of v
19: **if** $sched$ is feasible **and** $sched$ has a TSL less than min_tsl **then**
20: $min\_tsl = TSL$ of $sched$
21: $best\_sched = sched$
22: **end if**
23: un-schedule v
24: **end if**
25: **end for**
26: implement $best\_sched$
27: **return**

References

Abdekhodaee A, Dunstall S, Ernst TA, Lam L (2004) Integration of stockyard and rail network: a scheduling case study. In: Proceedings 5th Asia-Pacific industrial engineering and management systems conference, Gold Coast, Australia, December 12–15, pp 1–16

Bay M, Crama Y, Langer Y, Rigo P (2010) Space and time allocation in a shipyard assembly hall. Ann Oper Res 179:57–76

Boland NL, Savelsbergh MWP (2012) Optimizing the Hunter Valley coal chain. In: Gurnani H, Mehrotra A, Ray S (eds) Managing supply disruptions: theory and practice of managing risk. Springer, London, pp 275–302

Conradie DG, Morison LE, Joubert JW (2006) Scheduling at coal handling facilities using simulated annealing. Math Method Oper Res 68:277–293

Cordeau J-F, Laporte G, Moccia L, Sorrentino G (2011) Optimizing yard assignment in an automotive transshipment terminal. Eur J Oper Res 215:149–160

Glassrock CD, Hoare RT (1995) An application of dynamic simulation to a bulk solids handling facility. In: Proceedings of the 5th international conference on bulk materials storage, handling and transportation, Barton, ACT Institution of Engineers, Newcastle, Australia, pp 291–297

Lee DH, Cao JX, Shi QX, Chen JH (2009) A heuristic algorithm for yard truck scheduling and storage allocation problems. Transp Res Part E 45:810–820

Liu SQ, Kozan E (2009) Scheduling trains as a blocking parallel-machine job shop scheduling problem. Computers & Operations Research 36:2840–2852

Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: a survey. Eur J Oper Res 141:241–252

Muller NZ, Mendelsohn R, Nordhaus W (2011) Environmental accounting for pollution in the united states economy. Am Econ Rev 101:1649–1675

Newman AM, Rubio E, Caro R, Weintraub A, Eurek K (2010) A review of operations research in mine planning. Interfaces 40:222–245

Ng WC, Mak KL, Li MK (2010) Yard planning for vessel services with a cyclical calling pattern. Eng Optimiz 42:1039–1054

Welgama PS, Oyston R (2003) Study of options to increase the throughput of the Hunter Valley coal chain. In: Proceedings of MODSIM 2003, Townsville, Australia, July 14–17, pp 1841–1846

Wong A, Kozan E (2010) Optimization of container process at seaport terminals. J Oper Res Soc 61:658–665