Scheduling with Fixed Maintenance, Shared Resources and Nonlinear Feedrate Constraints: a Mine Planning Case Study

Christina N. Burt, Nir Lipovetzky, Adrian R. Pearce, and Peter J. Stuckey

Department of Computing and Information Systems,
The University of Melbourne, Parkville Australia

Abstract. Given a short term mining plan, the task for an operational mine planner is to determine how the equipment in the mine should be used each day. That is, how crushers, loaders and trucks should be used to realise the short term plan. It is important to achieve both grade targets (by blending) and maximise the utilisation (i.e., throughput) of the mine. The resulting problem is a non-linear scheduling problem with maintenance constraints, blending and shared resources. In this paper, we decompose this problem into two parts: the blending, and the utilisation problems. We then focus our attention on the utilisation problem. We examine how to model and solve it using alternative approaches: specifically, constraint programming, MIQP and MINLP. We provide a repair heuristic based on an outer-approximation, and empirically demonstrate its effectiveness for solving the real-world instances of operational mine planning obtained from our industry partner.

1   Introduction

In open-pit mines, a common form of materials handling is through truck and loader fleets [16], where the loaders excavate the material from blocks and the trucks haul it to dumpsites, stockpiles, run–of–mine (rom) stockpiles, or directly to the crusher. In this paper, we will consider a challenging scheduling problem that arises in the context of this form of materials handling. We denote the movement of material by a *movement*, which is a representation of the material, its source location (a block or stockpile), its destination location (a crusher or stockpile) and the grade of material. In Figure 1, we represent the *movements* of material by edges. Our task is to schedule these movements subject to constraints on the plants and equipment. At most one loader may excavate a movement. Since loader traversal is slow ($\approx$5km/h), it is preferable to sequence the loader's tasks in such a way that loader traversal doesn't prohibit flow of material to the crusher. That is, at least one loader should always be working at any given time. Thus, we can think of the task of *sequencing the loaders* as sequencing the movements. The loaders transfer material to the trucks, which haul the material to one of various destinations. Importantly, the trucking resources are limited and *shared* between all loaders. The material flow to the crusher is key to measuring productivity of the mine, and therefore it is important to *maximise the feedrate* to the crusher at any moment in time.
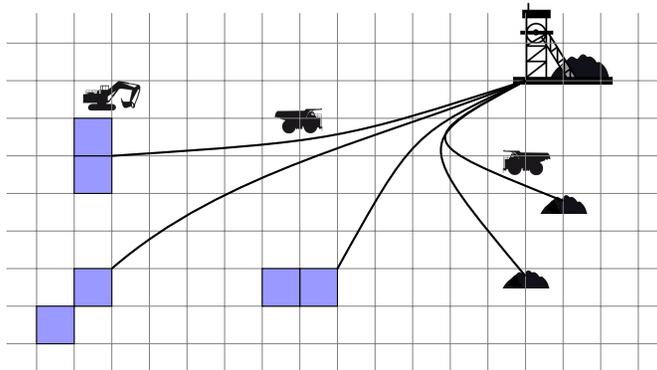
Fig. 1. An abstract representation of a mine. The squares are *movement* sources; the edges are *movements*; *loaders* excavate the movements at the source; *trucks* haul material from the loaders to the *crusher* or the various *stockpiles*. Icon images adapted from [4] and [8].

In open-pit mining operations, there are several levels of planning, each of which pass down restrictions in decisions. That is, long-term plans (strategic) are handed down to short-term planners, who in turn generate plans for operations planners (tactical). At each of these levels of planning, the task is to determine the order in which material *movements* should be mined and how they should be processed, such that blending and utilisation targets are met. Additionally, all of the equipment, including crushers, are subject to *maintenance tasks* which create periods of unavailability. The operations planners have the task of enacting the plans—physically, with trucks and loaders—such that the mine performance goals are met. At the core of our work is a nonlinear scheduling problem with shared resources, blending, and maintenance constraints. The goals of interest in our research are (a) correct blending of materials, and (b) utilisation of equipment. The output of a tactical mine plan is a sequence of *builds* (i.e., small, short-term stockpiles) with an allocation of partial movements to builds such that the builds have the correct blend and the crusher is maximally utilised.

In this paper, we investigate principled approaches for both modelling and solving a subproblem of tactical mine planning, which requires flexible partitioning of tasks to facilitate *fixed maintenance tasks*. We arrive at the subproblem by first decomposing the global problem into the blending and utilisation problems. We then derive *event-based* formulations for the latter subproblem: exploring formulations in constraint programming and mathematical programming. Motivated by finding a more computationally efficient solving approach, we derive an outer-approximation and repair heuristic which utilises aspects from each of our formulations.

The contributions of this paper are:

- a new modelling and solution approach for an operational mine scheduling problem with flexible partitioning, that allows for fixed maintenance tasks;

- a comparison of modelling approaches for this problem, using CP and OR techniques;
- a repair heuristic for obtaining efficient solutions to the subproblem; and
- a detailed set of experiments on real-world instances of the problem.

In previous work [12], we consider a different version of the build planning problem. The model in [12] does not consider maintenance, and does not include the crusher requirements. These considerations were considered crucial by our industry partner, and lead to a nonlinear problem. Another critical difference is that the modelling approach to the build planning problem in this paper uses CP and MIP/MINLP technology, whereas in [12] we use planning technology to account for state-dependent components that we do not include in this paper. Overall, this paper results from moving to a more realistic version of the problem we addressed in [12].

In the literature, scheduling problems with fixed maintenance have been addressed for up to 2 machines [1, 11, 15] and m machines [13, 9]. None of these works consider the side constraints of shared resources, blending or feedrate constraints. The latter, [9], does consider a nonlinear objective function and provides a linearisation of the model. In [10], the authors consider an integrated scheduling problem, which is equivalent to the loader sequencing problem without maintenance tasks. However, it is clear from the scheduling with maintenance literature that the problem we consider here has not been addressed.

In the mining literature, the shared resources and blending constraints have been addressed. In particular, [2] develop an outer-approximation and repair heuristic for nonlinear blending constraints at stockpiles. This problem is the most similar to ours from the literature that we could find. However, as these authors consider a longer planning horizon, they do not consider the traversal of equipment or the feedrates at the crushers. The key difference between the nonlinear blending at stockpiles and the nonlinear feedrate at the crushers is that it is not possible to obtain tight bounds on the feedrate at the crushers. Apart from this, the nonlinear constraints that arise are both bilinear in form.

The remainder of the paper is organised as follows. We first describe the subproblem that addresses the problem of scheduling the equipment and sequencing the movements to keep the crushers busy. We formulate this problem using event-based models, allowing for mixed fleets, in Section 3. Then, we extend these formulations to account for maintenance constraints in Section 4. We provide a heuristic approach based on outer-approximation and feasibility repair in Section 5. We provide experiments across all models in Section 6. We conclude with a discussion of our experience in Section 7.

2 Problem Description

For our industry partner, operations planning and grade control—which together form the tactical mine scheduling problem—are performed separately. The grade controller (at the build) is interested in minimising deviation of the grade *blend*. Once the grade control plan is constructed we need to solve the utilisation problem to complete it as efficiently as possible.

In this sense, if there are multiple pits in a mine, each with their own crusher, then it is important that the scheduled tasks are aligned, i.e. complete at the same time. That is, there may be multiple crushers that create one build at the end of the supply chain. Therefore, these subproblems within pits (for example) are not completely independent.

Problem Description 1 (Tactical mine scheduling). *Given a set of movements to be mined, determine the mining sequence such that the movements form builds with correct blend bounds, the crushers contributing to a build are aligned, and the utilisation of the crusher is maximised.*

In practical applications, the sequence must respect *movement precedences*, which determine which movement is accessible first. For example, it may be required to clear all movements associated with a particular location before another movement (or set of movements) is possible. Additionally, all of the equipment—trucks, loaders and crushers—are subject to *maintenance* at pre-defined periods. The sequencing should therefore also account for maintenance of equipment.

In previous work (see [12]), we decomposed this problem into the blending and utilisation problems. In this paper, we will adopt the same scheme with the additional requirement that we align the crusher tasks contributing to the same build. We then focus completely on the utilisation problem in the remainder of this paper. For a general description of the decomposition approach, see [3].

In our decomposition approach, we partition the problem into two subproblems. We first model the blending subproblem as a mixed-integer program. We add new constraints that approximate the time required to mine a build, for each crusher, in the context of a mine with multiple crushers. To achieve this, we ensure that the contribution (in tonnes) of each crusher to the build, is proportional to the maximum feedrate of the crusher itself. That is:

$$\frac{C_i}{\bar{\Phi}_i} = \frac{C_j}{\bar{\Phi}_j},$$

where C_i is the total quantity of material contributed to the build from crusher i, and $\bar{\Phi}_i$ is the maximum feedrate of crusher i. To improve robustness, we introduce these alignment constraints in the objective function: i.e., as soft constraints. The output of this first partition is an allocation of movements to builds such that the blending constraints are met, and the builds are aligned with respect to maximum crusher utilisation.

The utilisation subproblem amounts to the question: how should the equipment be used so that our mining goals are met and the equipment is maximally utilised? The components of this problem include:

1. **sequencing the loaders**—this allows the loader traversal times between movements to be counted, and allows a feedrate to be allocated to each movement.
2. **allocating the trucks**—the cycle time (i.e., round trip travel time between pick-up and dumping locations and back) is accounted for.
3. **maximising feedrate at crusher**—the incoming ore feedrate is limited by the capacity of the crusher, and yet the crusher should not be scheduled to be underutilised.

And, key to this problem, we must allow for maintenance events. We assume that we have obtained a sub-set of the movements from a solution to the blending subproblem, such that the blending constraints are already met.

Problem Description 2 (Build Planning). *Given a set of movements that together meet blending constraints, determine the mining sequence that maximises the crusher utilisation, such that loading and truck fleet capacity constraints are met, and maintenance events are accounted for.*

The underlying challenges in this problem arise from the essential non-linearity of determining the mining rate (tonnes/minute) of a movement m. The feedrate of a movement, ϕ_m, is determined by the capacity, C_t^T, of each truck, t, assigned to the movement divided by the cycle time of the truck travelling from movement source to the crusher and back again, $T_{m,t}^T$. Conversely, the duration of mining the movement is the ratio of the size of the movement to the movement feedrate. Summing up the feedrates of the movements currently being sent to the crusher will determine the current crusher utilisation. Summing up the trucks assigned to all movements currently mined must remain below the trucking capacity limit. Hence we have a scheduling problem with variable durations and resource usages where we are trying to maximise utilisation of resources.

The fact that the movements themselves are discrete leads to an intuitive discretisation of the problem. That is, it is intuitive to assign a feedrate and truck allocation to each movement. However, this is actually an unnatural *restriction* on the problem—it might be better to, for example, change the truck allocation part way through mining the movement. This is not only important for optimising the solution (or finding feasible solutions in very tightly constrained instances), but also for accounting for maintenance.

With maintenance events, it is possible that the discretisation will lead directly to poor quality solutions in terms of crusher utilisation. Consider, for example, the case where the durations between maintenance events are so small that the given movement sizes cannot be mined, subject to operational constraints. In this case, it would be ideal to determine ways to partition the movements such that good solutions can be obtained.

3 Build Planning Models

We begin by addressing the problem of scheduling the crusher without maintenance tasks. In this section, we formalise our approaches as event-based models. This is, in part, inspired by Automated Planning encodings, and in part by the logical representation of Constraint Programming.

3.1 Discretised Approach: Constraint Programming

We define the build planning problem by a set of movements, $m \in \mathcal{M}$, with a defined size (in tonnes), C_m^M, a source location, O_m, a destination location, D_m, and precedences between movements, $p \in \mathcal{P}$, defined by $p = (m', m) \in \mathcal{M} \times \mathcal{M}$ that require a specific movement, m', is completed before another, m, can begin. Additionally, the problem has a set of loaders, $l \in \mathcal{L}$, a set of crushers, $\kappa \in \mathcal{F}$, a set of truck types, $t \in \mathcal{T}$, and the number of trucks of each type, N_t^T.

For each crusher we have a maximum feed rate, $\bar{\Phi}_\kappa$. For each loader we have a maximum dig rate, $\bar{\Phi}_l^L$, and a time to traverse from the source location of movement m to the source location of movement m', $T_{m,m',l}^L$. For each truck type, we have a truck capacity, C_t^T, and a cycle time for each movement m, $T_{m,t}^T$—that is, how long it takes the truck to go from the source location O_m to the destination D_m and back to O_m. In addition, we assert a maximum number of movements per loader, N_l^M.

The principal decisions to be made are:

- for each loader l, the sequence of movements $d_{l,i} \in \mathcal{M} \cup \{\bot\}, i = 1, \ldots N_l^M$ it will complete (with $\bot$ representing dummy unused movements),
- for each movement, $\tau_{m,t}$ is the number of trucks of different types, $t \in \mathcal{T}$, assigned to the movement.

Auxiliary variables give:

- the start, s_m, duration, t_m, and end times, e_m, for each movement,
- the movement rate, ϕ_m, (tonnes/minute) for each movement,
- the loader assigned to each movement, λ_m,
- indicator variables, $z_{m,m'}^\wedge$, for which movements m are running when movement, m', is started.

The constraints are:

- Precedence constraints are satisfied:
$$e_m \leq s_{m'}, \qquad\qquad (m, m') \in \mathcal{P}. \tag{1}$$

- Dummy movements $\bot$ are at the end
$$d_{l,i} = \bot \rightarrow d_{l,i+1} = \bot, \qquad l \in \mathcal{L}, i \in 1 \ldots N_l^M - 1. \tag{2}$$

- The next task cannot begin until the loader has moved there:
$$e_{d_{l,i}} + T_{m,m',l}^L \leq s_{d_{l,i+1}}, \qquad l \in \mathcal{L}, i \in 1 \ldots N_l^M - 1, d_{l,i+1} \neq \bot. \tag{3}$$

- Each movement is assigned to at most one loader (assuming $\bot = 0$):
$$\mathtt{all\_different\_except\_0}([d_{l,i} \mid l \in \mathcal{L}, i \in 1 \ldots N_l^M]). \tag{4}$$

- Ensure the λ_m and $d_{l,i}$ variables agree:
$$d_{l,i} = m \rightarrow \lambda_m = l, \qquad\qquad m \in \mathcal{M}, l \in \mathcal{L}, i \in 1 \ldots N_l^M, \tag{5}$$
$$\lambda_m = l \rightarrow \exists_{i \in 1 \ldots N_l^M} d_{l,i} = m. \tag{6}$$

- The feedrate for a movement is constrained by loader dig rate:
$$\phi_m \leq \bar{\Phi}_{\lambda_m}^L, \qquad\qquad m \in \mathcal{M}. \tag{7}$$

- Movement rate is constrained by trucking capacity assigned:
$$\phi_m \leq \sum_{t \in \mathcal{T}} \tau_{m,t} \times C_t^T / T_{m,t}^T, \qquad m \in \mathcal{M}. \tag{8}$$

- Duration of a movement is given by the tonnage divided by move rate:

$$t_m = \frac{C_m^M}{\phi_m}, \qquad\qquad m \in \mathcal{M}. \tag{9}$$

- Start and end times are related by movement duration:

$$s_m + t_m = e_m, \qquad\qquad m \in \mathcal{M}. \tag{10}$$

- Crusher is not overloaded:

$$\sum_{m \in \{\mathcal{M}|D_m=\kappa\}} z^{\wedge}_{m,m'}\phi_m + \phi_{m'} \le \bar{\Phi}_\kappa, \quad \kappa \in \mathcal{F}, m' \in \{\mathcal{M} \mid D_{m'} = \kappa\}, \tag{11}$$

  We only test overload at the start time of any movement, since that is the only time when more can be fed to a crusher.
- Trucking capacities are respected:

$$\texttt{cumulative}(s_m, t_m, [\tau_{m,t}|m \in \mathcal{M}], N_t^T), t \in \mathcal{T}. \tag{12}$$

- Indicator variables for coinciding events are correct:

$$z^{\wedge}_{m,m'} \leftrightarrow (s_m \le s_{m'} \wedge e_m > s_{m'} \wedge m \ne m'). \tag{13}$$

The objective function is to minimise the makespan, i.e. minimise $\max_{m \in \mathcal{M}} e_m$. Our search strategy first assigns a movement to each loader, $d_{l,i} \in \mathcal{M}$, then assigns the earliest possible start times, s_m, and tries to assign the maximum feedrate, ϕ_m, while minimising the number of trucks assigned, $\tau_{m,t}$.

3.2 Discretised Approach: Mixed-integer Quadratic Programming

For a mathematical programming approach, we wish to keep the event-based representation and linearise the constraints as much as possible. The loader sequencing problem is represented by a graph where nodes are movement source locations, and edges are traversals of loaders from one movement source to another. Loader traversal decisions are represented by flow variables, $x_{m,m',l}$, which take a non-zero integer value if loader l performs movement m followed directly by m'. We extend the movement set $\mathcal{M}$ to include a dummy source, σ, and sink, σ', such that $\mathcal{M} \cup \{\sigma, \sigma'\} = \mathcal{M}'$.

We encode the loader sequences constraints with a node-disjoint multi-commodity flow formulation, which effectively allocates loaders to movements and derives their traversal sequence. That is,

$$\sum_{m' \in \mathcal{M}', l} x_{m',m,l} - \sum_{m' \in \mathcal{M}', l} x_{m,m',l} = \begin{cases} \min\{|\mathcal{M}|, |\mathcal{L}|\} & \text{if } m = \sigma, \\ -\min\{|\mathcal{M}|, |\mathcal{L}|\} & \text{if } m = \sigma', \\ 0 & \text{otherwise,} \end{cases} \quad \forall\, m \in \mathcal{M}, \tag{14}$$

$$\sum_{l, m' \in \mathcal{M}'} x_{m',m,l} = 1 \qquad\qquad \forall\, m \in \mathcal{M}', \tag{15}$$

$$e_{m'} + \sum_l T^L_{m',m,l} x_{m',m,l} - MS(1 - \sum_l x_{m',m,l}) \le s_m \qquad \begin{aligned} &\forall\, m' \in \mathcal{M}' \\ &m \in \mathcal{M} \\ &m \ne m', \end{aligned} \tag{16}$$

where MS represents the maximum possible makespan. In constraint (14), we create sequences for each loader. However, if the number of blocks is less than the number of loaders, we must restrict the sequences to the smaller number. Constraint (15) ensures that all blocks are visited by exactly one loader. We use a big-M formulation for constraint (16) to ensure the travel time for each loader is accounted for, but only if that edge is traversed by that loader.

We can now use the flow variables to indicate which loader performs a task— this permits us to encode the loader maximum dig-rate bounding the movement feedrate:

$$\phi_m \leq \sum_{l,m'} \bar{\Phi}_l^L x_{m',m,l} \qquad \forall \, m \in \mathcal{M}. \tag{17}$$

To encode the `cumulative` constraints we must determine whether two events coincide. To do this, we reason that, for any two events, if they both end after the other began, then the two events coincide. To formalise this, we introduce binary variables $z_{m,m'}^{\succ}$ to *indicate* that movement m finishes after m' starts. Recall that $z_{m,m'}^{\wedge}$ indicates that event m occurs at the same time as the start of event m'. The constraints to activate these variables are:

$$MSz_{m,m'}^{\succ} \geq e_m - s_{m'} \qquad \forall \, m \in \mathcal{M}, m' \in \mathcal{M}, \tag{18}$$

$$z_{m,m'}^{\wedge} \geq z_{m,m'}^{\succ} + z_{m',m}^{\succ} - 1 \qquad \forall \, m \in \mathcal{M}, m' \in \mathcal{M}, \tag{19}$$

$$z_{m,m'}^{\wedge} \leq \frac{z_{m,m'}^{\succ} + z_{m',m}^{\succ}}{2} \qquad \forall \, m \in \mathcal{M}, m' \in \mathcal{M}. \tag{20}$$

The `cumulative` trucking capacity constraints can now be encoded as

$$\sum_{m \in \mathcal{M} \setminus \{m'\}} \tau_{m,t} z_{m,m'}^{\wedge} + \tau_{m',t} \leq N_t^T \qquad \forall \, m' \in \mathcal{M}, t \in \mathcal{T}. \tag{21}$$

The crusher feed rate constraints are similarly encoded as

$$\sum_{m \in \{\mathcal{M} \setminus \{m'\} | D_m = \kappa\}} \phi_m z_{m,m'}^{\wedge} + \phi_{m'} \leq \bar{\Phi}_\kappa \qquad \forall \, m' \in \{\mathcal{M} | D_{m'} = \kappa\}. \tag{22}$$

With the exception of the nonlinear duration calculation, the remaining constraints as presented in the constraint programming model are linear, and can be used directly in the mathematical programming model. The constraints (22) and (21) can be linearised exactly, leaving us with a mixed-integer quadratic programming formulation with positive semi-definite form of constraint (9).

We linearise (21) and (22) by introducing two ancillary variables $\phi'_{m,m'}$ and $\tau'_{m,m',t}$, which will take on the value of ϕ_m if $z_{m,m'}^{\wedge}$ is 1, and zero otherwise. Let $\bar{\Phi}$ and $\bar{\tau}_m$ be the upper bounds on their respective variables. Then,

$$\phi'_{m,m'} \geq \phi_m - (1 - z_{m,m'}^{\wedge})\bar{\Phi} \qquad \forall \, m, m' \in \{\mathcal{M} \mid D_m = \kappa, D_{m'} = \kappa\}, \tag{23}$$

$$\phi'_{m,m'} \leq \phi_m + (1 - z_{m,m'}^{\wedge})\bar{\Phi} \qquad \forall \, m, m' \in \{\mathcal{M} \mid D_m = \kappa, D_{m'} = \kappa\}, \tag{24}$$

$$\tau'_{m,m',t} \geq \tau_{m,t} - (1 - z_{m,m'}^{\wedge})\bar{\tau}_m \qquad \forall \, m, m', m \neq m', t, \tag{25}$$

$$\tau'_{m,m',t} \leq \tau_{m,t} + (1 - z_{m,m'}^{\wedge})\bar{\tau}_m \qquad \forall \, m, m', m \neq m', t. \tag{26}$$

Then, we alter constraints (21) and (22) as follows:

$$\sum_{m \in \mathcal{M} \backslash \{m'\}} \tau'_{m,m',t} + \tau_{m',t} \leq N_t^T \qquad \forall\, m' \in \mathcal{M}, t \in \mathcal{T}, \tag{27}$$

$$\sum_{m \in \{\mathcal{M} \backslash \{m'\} | D_m = \kappa\}} \phi'_{m,m'} + \phi_{m'} \leq \bar{\Phi}_\kappa \qquad \forall\, m' \in \{\mathcal{M} | D_{m'} = \kappa\}. \tag{28}$$

We improve computational performance with the following valid inequalities:

$$\sum_{m,m' \in \mathcal{M}'} x_{m',m,l} \leq 1 \qquad\qquad \forall\, l \in \mathcal{L}, \tag{29}$$

$$\sum_{m' \in \mathcal{M}',l} x_{m',m,l} = 1 \qquad\qquad \forall\, m \in \mathcal{M}, \tag{30}$$

$$\sum_{m' \in \mathcal{M}',l} x_{m,m',l} = 1 \qquad\qquad \forall\, m \in \mathcal{M}, \tag{31}$$

$$e_m \leq MS \sum_{m' \in \mathcal{M}',l} x_{m',m,l} \qquad \forall\, m \in \mathcal{M}, k \in \mathcal{K}, \tag{32}$$

$$z^\wedge_{m,m'} = z^\wedge_{m',m} \qquad\qquad \forall\, m \in \mathcal{M}, m' \in \mathcal{M}'. \tag{33}$$

In the objective function and constraint (34), we represent the makespan as a `max` function over all movement event end times. Thus we obtain a model of the discretised heterogeneous crusher scheduling problem as follows:

$$MIQPDisc: \qquad \min \quad \omega$$
$$\text{s.t.} \quad \omega \geq e_m \qquad\qquad \forall\, m \in \mathcal{M}, \tag{34}$$
$$(1), (8)\text{--}(10), (14)\text{--}(20), (23)\text{--}(33),$$
$$z^\wedge_{m,m'}, z^\succ_{m,m'} \in \{0,1\},$$
$$x_{m,m',l}, \phi'_{m,m'}, \tau_{m,t}, \tau'_{m,m',t}, \phi_m, t_m, \omega, s_m, e_m \in \mathcal{R}^+.$$

3.3 Overview: Discretised Approach

As we will see in the experiments section (Section 6), the models we presented solve easily in constraint programming, mixed-integer nonlinear programming and mixed-integer quadratic programming solvers for realistic sized instances. One key issue with this approach, however, is that the trucking fleet is allocated to one movement for the entirety of the mining event. This is equivalent to fixing the feedrate for the entire event. On one hand, this alone can lead to poor crusher utilisation. On the other hand, we wish to introduce maintenance tasks, which too can lead to poor crusher utilisation. For example, consider a scheduling problem with one loader and one movement. Suppose the minimum duration of the event is 1000, and the maintenance task occurs between $999 \leq t \leq 1099$. This will force the makespan to be 2099, while the crusher is doing nothing for the first 1099 time units.

This strongly motivates a need to be able to *partition* the movements on-the-fly. In the following section, we will extend our models to allow for this type

of flexible partitioning. For simplicity, we present only the loader maintenance constraints, from which it is straightforward to extend the model to account for truck and crusher maintenance.

4 Build Planning Models with Flexible Partitioning

In this section, we extend the formulations from previous sections by introducing a flexible partitioning of each movement. We restrict our formulation to the case of two partitions. However, further partitions are an easy extension, but with particular attention paid to the symmetry constraints. The flow constraints are sufficient in the form of constraints (14)–(15). The variables s_m, e_m, $\tau_{m,t}$, $z^{\succ}_{m,m'}$, $z^{\wedge}_{m,m'}$, ϕ_m and t_m extend with an additional index representing the partition.

The new partitions have unknown size. Therefore, we require a variable, $c_{m,k}$, to represent the size of the partition (in tonnes), where the total size must equal the original size of the movement:

$$\sum_k c_{m,k} = C_m \qquad\qquad \forall\, m \in \mathcal{M}. \tag{35}$$

Importantly, this leads to an expression for feedrate that is no longer positive semi-definite:

$$\phi_{m,k} = \frac{c_{m,k}}{t_{m,k}} \qquad\qquad \forall\, m \in \mathcal{M}, k \in \mathcal{K}. \tag{36}$$

Symmetry is a big issue when we can partition a movement anywhere and then alternate the order of the partitions. To restrain this computational issue, we introduce the following constraint (in combination with a restriction on k and $k+1$ in constraint (38)):

$$s_{m,k+1} \geq e_{m,k} \qquad\qquad \forall\, m \in \mathcal{M} \quad (37)$$
$$k < |\mathcal{K}| - 1,$$

$$e_{m',k+1} + \sum_l T^L_{m',m,l} x_{m',m,l} - MS\left(1 - \sum_l x_{m',m,l}\right) \leq s_{m,k} \quad \forall\, m \in \mathcal{M} \quad (38)$$
$$m' \in \mathcal{M}'$$
$$m \neq m'$$
$$k < |\mathcal{K}| - 1.$$

We first ensure that the $k+1$th partition follows the kth partition (of the same movement) with respect to start time—see constraint (37). Then, we ensure that only the first partition includes the loader traversal time.

We incorporate maintenance tasks for the loaders into the partition model as follows. Each maintenance task has a predefined start (s^H_l) and finish (e^H_l) time. We require that each maintenance task does not overlap with the scheduled tasks

$$e_{d_{l,i},k} \leq s^H_l \vee s_{d_{l,i},k} \geq e^H_l \vee d_{l,i} = \bot, \quad \forall\, l \in \mathcal{L}, k \in \mathcal{K}, i \in 1 \ldots N^M_l. \tag{39}$$

We can encode this for MIP models by introducing variables indicating a movement has finished before, $z^{H,\prec}_{m,k,l}$, or after, $z^{H,\succ}_{m,k,l}$, each maintenance task l, and use a big-M approach as follows:

$$e_{m,k} \leq s_l^H + (1 - z_{m,k,l}^{H,\prec})MS + (1 - \sum_{m'} x_{m',m,l})MS \quad \forall\, m,k,l, \quad (40)$$

$$s_{m,k} \geq e_l^H - (1 - z_{m,k,l}^{H,\succ})MS - (1 - \sum_{m'} x_{m',m,l})MS \quad \forall\, m,k,l, \quad (41)$$

$$\sum_{m'} x_{m',m,l} = z_{m,k,l}^{H,\prec} + z_{m,k,l}^{H,\succ} \quad \forall\, m,k,l. \quad (42)$$

Thus we obtain the following mixed-integer nonlinear program:

$$MINLPexact: \qquad \min \quad \omega$$

s.t. $\quad (1)^*, (8)^*, (10)^*, (14)^*-(15)^*, (17)^*-(20)^*, (34)^*, (35)-(38), (40)-(42),$

$$z_{m,k,l}^{H,\prec}, z_{m,k,l}^{H,\succ}, z_{(m,k),(m',k')}^{\wedge}, z_{(m,k),(m',k')}^{\succ} \in \{0,1\},$$

$$\tau_{m,k,t}, x_{m,m',l}, \phi_{m,k}, \phi_{(m,k),(m',k')}', c_{m,k}, \tau_{(m,k),(m',k')}', t_{m,k}, \omega, s_{m,k}, e_{m,k} \in \mathcal{R}^+.$$

Constraints marked with $(*)$ are extended to account for partitions in the obvious way.

Since the partition is flexible, it seems plausible that the MINLP representation provides an optimal solution to the utilisation subproblem. However, this is not the case, as we show in the following theorem.

Theorem 1. *Let m_1 be a movement that can be flexibly partitioned, and let all remaining movements, $\{\mathcal{M}\backslash m_1\}$, be fixed such that they cannot be partitioned. Further, let maintenance tasks only exist for loaders. Then, the minimum number of partitions required for m_1 to guarantee a solution optimality for the overall problem is*

$$|\mathcal{K}_{m_1}| \geq \sum_l |Maint(l)| + |\mathcal{M}|.$$

Proof. Suppose there are $\mathcal{L} > 1$ loaders operating and there are no maintenance tasks. W.l.o.g, let l_1 mine only movement m_1 during the makespan of the build, and the remaining loaders mine the remaining $|\mathcal{M}| - 1$ movements. Since the remaining movements are fixed, if we consider $|\mathcal{M}|$ partitions, then we have considered all possible $|\mathcal{M}| - 1$ event end times, and therefore guarantee optimality. Now suppose each loader, l, has its own set of maintenance tasks, $Maint(l)$. Since each task may introduce a new event *partition* end time, we must consider a further $\sum_{\substack{i \in Maint(l) \\ \forall l \in \mathcal{L}}} i$ partitions in order to guarantee optimality. $\qquad\square$

Furthermore, when we allow all movements to partition, then the number of partitions (per movement) to guarantee optimality depends on the number of partitions introduced in all the movements. That is, there is a recursive relationship. The take-home message here is that the number of partitions required cannot be determined *a priori*, and therefore should be determined during search.

5 Outer-Approximation and Repair Heuristic

We can obtain a completely linear outer-approximation by introducing McCormick inequalities to represent the bilinear term $\phi_{m,k}\, t_{m,k}$. While a solution to

this model is not feasible for the original problem, it may provide us with useful information, such as a suggestion of the partitioning point for the movements. This gives rise to an Outer-Approximation and Repair Heuristic.

We approximate the bilinear term $\phi_{m,k}\, t_{m,k}$, using the upper $(\bar{\Phi}, \bar{T}_m^M)$ and lower $(\underline{\Phi}, \underline{T}_m^M)$ bounds on the variables, with constraints analogous to (23)–(26) to arrive at an ancillary variable, $\mu_{m,k}$. We substitute the ancillary variable into our feedrate constraint:

$$\mu_{m,k} \geq c_{m,k} \qquad\qquad \forall\, m,k. \qquad\qquad (43)$$

The outer-approximation model is therefore:

$$MIPouter: \qquad \min \quad \omega$$

s.t. $(1)^*, (8)^*, (10)^*(14)^*\text{–}(15)^*, (17)^*\text{–}(20)^*, (23)^*\text{–}(26)^*, (34)^*, (35)\text{–}(42), (43),$

$$z^{\wedge}_{(m,k),(m',k')}, z^{\succ}_{(m,k),(m',k')} \in \{0,1\},$$

$$\tau_{m,k,t}, \mu_{m,k}, \phi_{m,k}, \phi'_{(m,k),(m',k')}, x_{m,m',l}, c_{m,k}, \tau'_{(m,k),(m',k')}, t_{m,k}, \omega, s_{m,k}, e_{m,k} \in \mathcal{R}^+.$$

We use this model to obtain new partition breakpoints of existing movements. Specifically, we take the partitioned movement capacity, $c_{m,k}$, solution and fix this variable. This reduces the problem to a form that can be solved using $MIQPDisc$. This translation requires the following steps. We set the number of movements to be equal to the number of active partitions (i.e., movement partitions with capacity greater than zero). All movements that are split have their new indexes saved in a split movement set, $\mathcal{M}^S$. We perform a translation on all data indexed by movements.

Algorithm 1 The outer-approximation and repair heuristic for partitioning movements in the presence of maintenance tasks.

1: Solve $MIPouter$ model using a MIP solver.
2: **if** Optimal or Feasible **then**
3:     Partition movements according to $c_{m,k}$ solution.
4:     Fix new $C_m = c_{m,k}$.
5:     Solve $MIQPDisc$ model using MIQP solver.
6: **end if**

This repair heuristic is guaranteed to find a feasible partition and will never provide a solution worse than $MIQPDisc$, however it is not guaranteed to find an improved solution. In fact, when there are no maintenance constraints, the $c_{m,k}$ variables are not driven to find good solutions by any mechanism and we expect to obtain solutions equivalent to those found in $MIQPDisc$. However, once maintenance tasks are added, the $c_{m,k}$ variables have a strong bound and will snap to the maintenance tasks. Therefore, we expect much better quality solutions from the repair heuristic for problems with maintenance.

The quality of the repair heuristic is bounded from below by an optimal solution to $MIQPDisc$. Furthermore, if the repair heuristic is run with the same number of partitions as $MINLPexact$, then the quality of the repair heuristic is bounded from above by an optimal solution to $MINLPexact$.

6 Experiments

We validated all models by cross-checking the objective values on a set of validation instances. We created a set of test instances by extending a set of real instances provided by our industry partner. The base set included three weeks of movements, to be subdivided into five builds per week across three pits. In the context of our experiments, the equipment in the three pits are independent; they have their own crusher and truck and loader fleets. We therefore have 45 instances in the base set. On average there are 4.6 blocks and 1.4 stockpiles per pit—the biggest pit containing 9 blocks. The blocks and stockpiles range in size from 1.2kt up to 90kt. One pit has a mixed fleet of 5 loaders (4 types) and 13 trucks (1 type), while the other two pits have a mixed fleet of 6 loaders (2 types), and 17 trucks (2 types). We extend every pit instance in the following ways:

- *Symmetry*: we double the number of loaders from 5 and 6 to 10 and 12;
- *Truck constrained*: we decrease trucks from 13 and 17, to 6 and 8.
- *Maintenance*: for every pit, we schedule maintenance for the complete loader fleet in a cascade, each for 8 hours, decreasing the available loaders by 1 at any time. To make the instances more sensible to these events, we decreased the number of loaders available to 3 in total.
- *Extreme Maintenance* as failure: for every pit, we schedule maintenance for the complete loader fleet for 1000 minutes, thus allowing the crusher to be fed only by stockpiles[1] if they are available.

We run experiments on the base set and all extension sets: thereby obtaining 225 instances. Each instance is tested with a time-out of 5 and 300 seconds. We solve the blending problem using Cplex version 12.6 with default settings—all instances solved within milliseconds, and therefore the solver required no intervention to improve computational efficiency. We solve $MIQPDisc$ and $MIPouter$ with Gurobi version 5.6.3, using tuned parameters `GomoryPasses` $= 0$ and `PrePasses` $= 2$. We tested the quadratic models using numerical stability settings (i.e., `Presolve` $= 0$, `FeasTol` $= 1e - 9$, `Quad` $= 1$), but found these had no impact on the validity of solutions. We model $MINLPexact$ using Pyomo version 3.5. We solve $MINLPexact$ with the Scip Optimization Suite version 3.1.0, using IpOpt version 3.11.8, coinHSL version 2014.01.10 [7] and Cplex version 12.6. We model the constraint programming (CP) approach using MiniZinc [14], and solve it using Gecode 4.2.1 [6]. None of the solvers reach the maximum memory allowed of 4GB. $MIQPDisc$ and $MIPouter$ were able to find a solution for all tested instances, CP failed solving only 2 instances of the *constrained* benchmark, $MINLPexact$ failed between 3 to 6 instances in all benchmarks when the time limit was set to 5 seconds.

In Table 1, we show the average (CPU) time for solving each instance, the average makespan of each build (M. [min]), and the average completion time difference of the slowest and fastest crusher within the same build (M. D.). $MIQPDisc$ is consistently the fastest solver in terms of CPU time, and achieves the best (lowest) average makespan in 5 out of the 8 tested benchmarks.

---

[1] Stockpiles have dedicated loaders and do not require a loader to be moved. In practice we extend our models to accommodate this restriction.

Table 1. When 5 or 300 is not followed by any tag, it refers to the original benchmark with their respective timeouts. Tags *sym, const, maint,* and *maint_e* stand for the symmetric, constrained, maintenance and extreme maintenance variations. *CPU* stands for average CPU time in seconds, *M [min]* stands for average Makespan in minutes, *M. D.* stands for the average build-crushers alignment difference in minutes.

Bench. Id	MIQdisc			MIPouter			MINLPexact			CP		
	CPU	M. [min]	M. D.	CPU	M. [min]	M. D.	CPU	M. [min]	M. D.	CPU	M. [min]	M. D.
‖5‖	**1.3**	**1699**	**284**	**1.3**	1709	288	4.9	2095	1557	3.7	1738	364
‖5‖sym‖	**0.7**	**1642**	**157**	1.2	1693	267	5.0	1771	972	3.9	1708	291
‖5‖const‖	**1.4**	**2405**	**1412**	1.7	**2405**	1420	5.2	2167	2169	4.2	2408	1853
‖300‖	**23.0**	1654	186	34.9	**1652**	**174**	235.0	1774	455	164.8	1713	308
‖300‖sym‖	**4.6**	**1642**	**157**	20.4	1653	178	240.8	1761	414	173.3	1699	272
‖300‖const‖	**44.3**	2369	**1393**	51.0	2368	**1393**	268.1	**2341**	1542	208.9	2357	1770
‖300‖maint‖	**26.7**	2011	**750**	45.2	2013	758	188.9	**1978**	779	–	–	–
‖300‖maint_e‖	**10.7**	2521	1263	62.0	2317	1139	152.0	**2277**	1049	–	–	–

MIPouter have a similar performance in terms of makespan quality, outperforming *MIQPDisc* significantly in the extreme maintenance benchmark, which benefits from the flexible partitioning. In the other benchmarks, if *MIPouter* is able to solve the problem optimally, its makespans are as good or substantially better than *MIQPDisc*. This is not reflected on average, as *MIPouter* performs worse than *MIQPDisc* in those instances where it only finds a primal solution in the allotted time. Similarly, *MINLPexact* outperforms all other solvers in both maintenance and constrained benchmarks, when 300 CPU seconds are allowed. We remark that the only solver that substantially benefits from the increase of maximum computation time is *MINLPexact*. Therefore, we tested *MIPouter* and *MINLPexact* with a timeout of 900 seconds, but it did not result in the same quality improvement that we observed by changing from 5 to 300 seconds, so we omit those results. The *CP* approach has a competitive performance with respect to *MIP* solvers, but is not the best solver in any benchmark. *CP* failed solving the flexible partition model, and therefore we did not run this model on the maintenance sets. In terms of crusher alignment (M.D.), clearly the best results are achieved in the original benchmarks with 300 seconds, and even better in the symmetric version where more loaders were available. Crusher alignment is strongly correlated to the success of each solver on achieving 100% crusher utilisation, as we assumed the crushers operated at 100% efficiency in the blending model to help allocate the right proportion of tons for each pit. Whenever the average crusher feedrate decreases, the alignment is likely to be harmed. In Table 2, we provide the crusher and truck utilisation statistics. The best crusher utilisation is achieved by *MIQPDisc* in the 300 CPU time symmetric benchmark, which coincides with the best alignment achieved in Table 1.

MIQPDisc is the best model, achieving a crusher utilisation factor up to 96.9%. The constrained and maintenance benchmark variations harm the ability to fully utilise the crusher. Note that in those variations, it is not possible to achieve a 100% utilisation. The constrained version harms most significantly the *CP* approach, while the best models in the constrained version are *MIQPDisc* and *MIPouter*. The best models to handle maintenance tasks are *MIPouter* and *MINLPexact*, achieving an improvement of 6% and 7% respectively over *MIQPDisc*. Again, this highlights the benefits of flexible partitions.

Table 2. When 5 or 300 is not followed by any tag, it refers to the original benchmark with their respective timeouts. Tags *sym, const, maint,* and *maint_e* stand for the symmetric, constrained, maintenance and extreme maintenance variations. *Cr. Util.* stands for crusher utilization in (%); *Tr. Util.* is truck utilization in (%).

Bench. Id	MIQdisc		MIPouter		MINLPexact		CP	
	Cr. Util.	Tr. Util.	Cr. Util.	Tr. Util.	Cr. Util.	Tr. Util.	Cr. Util.	Tr. Util.
‖5‖	**94.3**	53.5	94.0	52.2	79.8	46.5	93.4	55.8
‖5‖sym‖	**96.9**	54.9	94.8	53.2	81.6	56.7	93.9	56.7
‖5‖const‖	**74.3**	75.5	**74.3**	75.4	71.5	59.9	67.8	32.6
‖300‖	96.3	55.0	**96.4**	54.2	91.7	56.2	94.0	56.3
‖300‖sym‖	**96.9**	55.3	96.2	55.3	91.8	59.5	94.2	56.8
‖300‖const‖	**75.1**	76.7	**75.1**	76.6	72.4	70.9	68.8	32.8
‖300‖maint‖	81.7	42.6	81.7	42.0	**83.7**	50.6	–	–
‖300‖maint_e‖	66.3	34.2	72.1	36.5	**73.1**	46.2	–	–

Loader utilisation is not shown in the table, as this resource is unconstrained in practice. We observe that in the data given to us by our industry partner, truck fleet availability is not too constrained either. As they pointed out, truck resources can become scarce in other data sets. To confirm this statement, we observed in the original benchmark that the most extremely truck constrained pit solved by *MIQPDisc* resulted in truck utilisation of 99.06% with 90% crusher utilisation, which highlights the ability of the solvers to push the truck utilisation to the maximum if it is required.

7 Discussion

An advantage of modelling the full problem using a high-level language, such as MiniZinc, is that it can lead to a more intuitive and natural representation of the problem. This simplified the extensions to a compact mathematical programming form, where the linearisations and logical constraints can sometimes be inelegant and cumbersome to de-bug. In this sense, it was beneficial to have multiple models from which we could validate the others results.

Beyond constraint programming and mathematical programming, the flexible partition scheduling problem could also be cast as a temporal planning problem over continuous variables with processes and events. We modelled this problem using the high-level *Planning Domain Description Language* (PDDL 2.1) [5], but no solver technology could handle the model. However, the flavour of planning encodings—event-driven modelling—is still present in our modelling approach.

Our industry partner solves this problem with an experienced planner and grade-controller, who develop plans by hand. They aim to achieve 80% utilisation at the crushers, after all practical constraints have been taken into account. While we cannot claim to have modelled all practical constraints, it it clear that these models will provide our partner with a useful decision-making tool with huge potential to push their crusher utilisation well beyond 80% where possible.

References

1. Aggoune, R.: Minimizing the makespan for the flow shop scheduling problem with availability constraints. European Journal of Operational Research 153(3), 534–543 (2004)
2. Bley, A., Boland, N., Froyland, G., Zuckerberg, M.: Solving mixed integer nonlinear programming problems for mine production planning with stockpiling. Tech. rep., University of New South Wales (2012), http://web.maths.unsw.edu.au/ froyland/bbfz.pdf
3. Burt, C.N., Lipovetzky, N., Pearce, A.R., Stuckey, P.J.: Approximate unidirectional benders decomposition. In: Proceedings of PlanSOpt-15 Workshop on Planning, Search and Optimization AAAI-15 (2015)
4. Coal Shovel Clip Art: Accessed: 17/11/2014 (2014), gofreedownload.net/
5. Fox, M., Long, D.: PDDL2. 1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
6. Gecode Team: Gecode: Generic constraint development environment (2006), available from `http://www.gecode.org`
7. HSL: (2013) a collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk
8. Immersive Technologies: Accessed: 17/11/2014 (2014), http://www.immersivetechnologies.com/
9. Jamshidi, R., Esfahani, M.M.S.: Reliability-based maintenance and job scheduling for identical parallel machines. International Journal of Production Research 53(4), 1216–1227 (2015)
10. Khayat, G.E., Langevin, A., Riopel, D.: Integrated production and material handling scheduling using mathematical programming and constraint programming. In: Proceedings CPAIOR (2003)
11. Kubzin, M.A., Strusevich, V.A.: Planning machine maintenance in two-machine shop scheduling. Operations Research 54(4), 789–800 (2006)
12. Lipovetzky, N., Burt, C.N., Pearce, A.R., Stuckey, P.J.: Planning for mining operations with time and resource constraints. In: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS-14 (2014)
13. Moradi, E., Ghoma, S.F., Zandieh, M.: Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem. Expert Systems with Applications 38(6), 7169–7178 (2011)
14. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming–CP 2007, pp. 529–543. Springer (2007)
15. Sbihi, M., Varnier, C.: Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness. Computers & Industrial Engineering 55(830–840) (2008)
16. Ta, C., Kresta, J., Forbes, J., Marquez, H.: A stochastic optimization approach to mine truck allocation. International Journal of Surface Mining 19, 162–175 (2005)