Scheduling of an underground mine by combining logic-based Benders decomposition and a priority-based heuristic *

Emil Lindh[1], Kim Olsson[1], and Elina Rönnberg[1][0000−0002−2081−2888]

Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden
elina.ronnberg@liu.se

Abstract. Underground mining is a complex operation that requires careful planning. The short-term scheduling, which is the scheduling of the tasks involved in the excavation process, is an important part of the planning process. In this paper, we propose a new method for the short-term scheduling of cut-and-fill mines.

Our problem formulation includes a new aspect of the problem, which is to handle that different excavation locations of the mine can have different priorities. The inclusion of this aspect allows the user to control the output of the scheduling and to direct resources to the locations where they are most needed according to the long-term plans. Our solution method consists of two components: a priority-based heuristic that constructs a complete solution by iteratively solving partial scheduling problems containing subsets of tasks, and a logic-based Benders decomposition scheme for solving these partial problems.

The computational performance of the proposed method is evaluated on industrially relevant large-scale instances generated from data provided by the mining company Boliden. Comparisons are made both with applying a constraint programming solver instead of the logic-based Benders decomposition scheme and with applying a constraint programming solver directly on the complete problem. The results show that our method outperforms the other two methods and yields schedules with a shorter makespan. The used instances are made publicly available to support further research in this area.

Keywords: Underground-mine scheduling · Cut-and-fill mining · Logic-based Benders decomposition · priority-based heuristic

* The collaboration with Torbjörn Viklund and Rasmus Tammia at Boliden has been vital for this project. They have contributed to discussing the problem formulation, constructing relevant instances, and evaluating the schedules. The work is partly funded by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05. Monolithic MIP and CP models for the studied problem were formulated and implemented in a student project by Daniel Cederberg, Jonatan Ek, Eric Felding, Emil Lindh, Kim Olsson, and Sam Olsson. The results from that project have been used for this paper with permission from the students.

1 Introduction

The excavation of an underground mine is a complex operation that requires careful planning. This planning is usually done in several phases with different time horizons. For an in-depth description of the planning phases, see [11]. The two components of the planning process that we address are the so-called *extraction plans* and the *short-term scheduling*. The extraction plans describe when a certain amount of ore from a certain part of the ore body should be excavated. The short-term scheduling then executes these plans by scheduling the activities and the machines involved in the excavation process.

In this paper, a new method is proposed for solving large-scale instances of a short-term scheduling problem for a cut-and-fill mine. Our problem formulation differs somewhat from previous work [11,13–15] as it includes a priority order between excavation locations. This extension of the formulation is proposed since, in practice, the urgency of the different excavation locations may differ. The reason for the differences can depend both on the nature of the extraction plan and on the progress made at each location, compared to what was expected. During excavation, activities are often postponed due to unforeseen events and the priority can therefore change between scheduling periods and it is important for the planners to be able to direct the resources to the locations where they are most needed.

Our solution method combines a priority-based heuristic with Logic-based Benders decomposition (LBBD). Computational results are provided for industrially relevant large-scale instances based on data from an operational mine. These instances have been made publicly available[1]. The project has been carried out in collaboration with the mining company Boliden and the paper is based on the master's thesis by the first two authors [6].

1.1 Problem definition

In cut-and-fill mining, the process of excavating a single volume of ore involves 11 tasks: *drilling*, *charging*, *blasting*, *ventilation*, *watering*, *loading*, *scaling*, *cleaning*, *shotcreting*, *bolting*, *facescaling* and *facecleaning*. In our problem formulation, the ventilation task is merged with the blasting task since none of them requires a machine; this results in having only 10 tasks to schedule. Together, these 10 tasks form an *excavation cycle*. A location where excavation takes place is called a *face*, and a mine can have several active faces in parallel.

Each task requires a specific type of machine and some tasks require the same machine type. There is a limited number of machines of each type and all machines of the same type are assumed to be identical. The tasks must be scheduled in the correct order at each face, and an excavation cycle cannot begin unless the previous cycle at the same face has been completed. Each day is divided into two work shifts, 06:30 - 14:30 and 15:30 - 00:00, and only the interruptible tasks can be started in one shift and finalised in the next. The

[1] https://gitlab.liu.se/eliro15/underground_mining_instances

shotcreting task requires a four-hour afterlag for the concrete to solidify before the next task can be performed. The blasting task can only occur during specific blasting windows between the work shifts, starting at either 15:00 or 00:42 and lasting 30 minutes. During a blasting window, the mine has to be evacuated due to safety reasons. Some tasks can be interrupted during the blasting windows while others cannot.

A task in a specific cycle that is to be scheduled at a specific face will henceforth be referred to as a *task instance*. Each machine type has an estimated velocity and when moving between two task instances at different faces, there is a travel time between the faces. The scheduling considers a number of excavation cycles at a number of faces and aims at minimising the sum of the individual makespans for the faces, while respecting the above-introduced constraints. A more in-depth problem description is found in [11] and [6].

1.2 Related work

The short-term scheduling problem considered in this work is not well-studied. There is however a series of papers from a PhD thesis [11] that addresses industrially relevant instances of underground mining problems. In this series of work, the scheduling of cut-and-fill mining is structured as a flow shop [13] and modelled by a CP formulation [14]. In [15], the problem formulation and the model of [13] and [14] are improved to better capture the characteristics of cut-and-fill mining, and the authors propose heuristics based on large neighbourhood search for solving the problem. The instances used in this series of work are unfortunately not available for further research or comparisons in this paper. Instead, we have made a re-implementation of their CP model to use for benchmarking. In [12], their results are extended to a more general underground-mining setting.

In [9] and [7], mixed-integer programming (MIP) approaches were applied to similar scheduling problems. In [9], the authors present a MIP model for a makespan minimizing mobile production fleet scheduling problem for a room-pillar mine. They solve small instances to optimality using a commercial solver and present heuristic methods for solving instances of larger sizes. In [7], the authors study a scheduling problem for a sublevel stoping mine, which includes the transportation of ore, and solve it using a MIP model and CPLEX. In [10], a scheduling support instrument is developed that schedules some of the activities in a production cycle.

To the best of our knowledge, there is no previous work where LBBD has been applied to this type of scheduling problem. However, LBBD has been successfully applied to other scheduling problems [1, 2, 5, 8].

1.3 Contributions and outline

Today, the short-term scheduling of the excavation process at Boliden is done manually. As manual scheduling is a time-consuming and complex operation that does not guarantee consistency in production efficiency, a more autonomous way of scheduling can yield great improvements. Heuristic methods for solving

realistic instances of this scheduling problem, with the objective to minimise makespan, have been developed [11]. However, many questions remain with respect to how to model and efficiently solve this problem. In particular, one crucial aspect for applicability in practice is the choice of objective function and how to take into account how the excavation progresses over time. Our work contributes to improved modelling of the problem and with a new strategy for solving it. These contributions are based on the following two fundamental observations about the problem that we have not seen addressed in previous work.

The first observation is that the problem has some inherent symmetries due to the machines being identical and the excavation cycles and the durations of tasks being the same for all faces. Even if the different travel times between faces contribute to breaking some of these symmetries, several solutions with the same or very similar makespan are likely to exist. A practical consequence of this property is that there is room for the decision-maker to choose between several plans with about the same efficiency in terms of length of makespan – which is the most important evaluation measure for a schedule.

Through discussions with practitioners, we learned that because of how the mining progresses and because of other aspects of the planning, the urgency of the faces can differ. This suggests that, in addition to the objective to minimise makespan, it is desirable to consider that there is a priority order between the faces. Therefore, we propose a priority-based heuristic to incorporate face priorities, while keeping the original makespan objective both in the models and as an evaluation measure for the solutions. Using this evaluation measure allows for a comparison with previous work. By using the priority-based heuristic, we both increase the decision-maker's control of the scheduling output and improve the computational times thanks to the symmetry breaking effect. This aspect of the model and method design is evaluated through a comparison between directly applying a CP solver using a carefully crafted model from previous work [11], referred to as M-CP, and applying an adaptation of this model in each iteration of the priority-based heuristic, referred to as H-CP.

The second observation is that by slightly reformulating the makespan objective, one obtains a formulation that lends itself to LBBD with a feasibility subproblem. This new objective was also developed in dialogue with Boliden. It is less granular than the standard objective function that directly aims to minimise the makespan and, instead, the focus is on which shifts to use to efficiently perform the tasks. Using the new objective, we introduce a LBBD scheme and tailored acceleration techniques to be applied in each iteration of the priority-based heuristic. The resulting approach is referred to as H-LBBD. Since the LBBD scheme is applied within the heuristic, its efficiency is evaluated in a direct comparison with approach H-CP.

The models and the heuristic are presented in Section 2 and the LBBD scheme, along with the acceleration techniques, is introduced in Section 3. Computational results are presented in Section 4. Conclusions and comments about possible future work are given in Section 5.

2 Modelling and priority-based heuristic

This section presents a complete CP model of the problem and the heuristic.

2.1 Monolithic CP-model

The CP model presented here is a re-implementation of the models presented in [11], and we refer to the previous work for a detailed motivation and description of the models. In previous work, the problem is described in a k-stage hybrid flow shop framework [13] where a job corresponds to an excavation cycle which is passed through k stages, each stage corresponding to a certain task within the excavation cycle. The model is adapted to the specific problem structure by including travel times and by removing time periods from the schedule in which the mine is evacuated or empty, i.e blasting windows and shift breaks. The latter results in a problem formulation in compressed time, and the solution is later post-processed to obtain the solution in the correct time. This compressed-time reformulation was introduced in [15], where a detailed description is found.

The notation used in the CP model is as follows. Let the set $\mathcal{M}$ index the machines, the set $\mathcal{F}$ index the faces, and the set $\mathcal{T} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ index the tasks. Further, let J be the maximum number of cycles to schedule and let the set $\mathcal{J}_f \subseteq \{1, \ldots, J\}$ index the cycles at face $f \in \mathcal{F}$. A task instance is specified by a task $t \in \mathcal{T}$ at a face $f \in \mathcal{F}$ in a cycle $j \in \mathcal{J}_f$. To simplify notation, we introduce a mapping of the indices of such task instance (tfj) to a single index a and let $\mathcal{A}$ contain all task instances. Let F_a denote the face at which task instance $a \in \mathcal{A}$ is to be scheduled. Furthermore, let the set $\mathcal{B}$ index the blasting windows b and let s_b be the time when blasting window $b \in \mathcal{B}$ starts and ends (these times coincides in the time-compressed model).

Let the set $\mathcal{M}_a \subseteq \mathcal{M}$ index the machines which can execute task instance $a \in \mathcal{A}$ and let the set $\mathcal{A}^{\mathrm{m}}_m$ index the task instances that machine $m \in \mathcal{M}$ can perform. Also, let the sets $\mathcal{A}^{(\mathrm{un})} \subseteq \mathcal{A}$, $\mathcal{A}^{(\mathrm{al})} \subseteq \mathcal{A}$, and $\mathcal{A}^{(\mathrm{bl})} \subseteq \mathcal{A}$ index the uninterruptible task instances, task instances that require afterlag, and blasting task instances, respectively. For each face $f \in \mathcal{F}$, let the ordered set $\mathcal{A}^{\mathrm{f}}_f$ index all the task instances, except the last one, in their execution order at face f.

Denote the duration of task instance $a \in \mathcal{A}$ by D_a and the duration of blasting window $b \in \mathcal{B}$ by $d^{(\mathrm{b})}$. Let the duration of the afterlag be denoted by $d^{(\mathrm{al})}$ and let the travel time between faces f, $f' \in \mathcal{F}$, be $l_{ff'}$. For each $a \in \mathcal{A}$ and $m \in \mathcal{M}_a$ the optional interval variable

$$I_{am} = (s_{am}, \mathbf{e_{am}}, D_a, \mathbf{o}_{am})$$

holds the start time s_{am}, end time $\mathbf{e_{am}}$, and duration D_a of task instance a. Also, it indicates by $\mathbf{o}_{am}$ whether machine m performs task instance a or not. To handle the compressed time, a variable $\mathbf{d}^{(\mathrm{al})}_a$ is introduced to calculate the correct afterlag. The variable $\mathbf{nextBlast}_{am}$ provides the next possible blast occasion following task instance $a \in \mathcal{A}$ executed by machine $m \in \mathcal{M}_a$. Lastly, a

precedence variable for a, $a' \in \mathcal{A}^{\mathrm{m}}_m$, $m \in \mathcal{M}$ is defined as

$$\mathbf{b}_{aa'} = \begin{cases} 1 & \text{if task instance } a \text{ precedes task instance } a' \text{ on machine } m, \\ 0 & \text{if task instance } a' \text{ precedes task instance } a \text{ on machine } m, \\ \perp & \text{if task instance } a \text{ and } a' \text{ are done by different machines.} \end{cases}$$

The complete CP-model is

$$\min \sum_{f \in \mathcal{F}} \max_{\substack{a \in \mathcal{A}^{\mathrm{f}}_f \\ m \in \mathcal{M}_a}} \Big(\mathbf{o}_{am} \times \mathbf{e}_{am} \Big),$$

$$\text{s.t.} \sum_{m \in \mathcal{M}_a} \mathbf{o}_{am} = 1, \quad a \in \mathcal{A}, \tag{1a}$$

$$s_{am} \in \{s_b | b \in \mathcal{B}\}, \quad a \in \mathcal{A}^{(\mathrm{bl})}, m \in \mathcal{M}_a, \tag{1b}$$

$$s_{am} \in \bigcup_{b \in \mathcal{B}} \{s_{b-1}, \ldots, s_b - D_a\}, \quad a \in \mathcal{A}^{(\mathrm{un})}, m \in \mathcal{M}_a, \tag{1c}$$

$$\mathbf{b}_{aa'} = 1 \iff$$
$$\mathbf{o}_{am} \wedge \mathbf{o}_{a'm} \wedge s_{am} + D_a + l_{F_a F_{a'}} < s_{a'm} \quad m \in \mathcal{M}, a, a' \in \mathcal{A}^{\mathrm{m}}_m, \tag{1d}$$

$$\mathbf{b}_{aa'} = 0 \iff$$
$$\mathbf{o}_{am} \wedge \mathbf{o}_{a'm} \wedge s_{a'm} + D_{a'} + l_{F_{a'} F_a} < s_{am}, \quad m \in \mathcal{M}, a, a' \in \mathcal{A}^{\mathrm{m}}_m, \tag{1e}$$

$$\mathbf{b}_{aa'} = \perp \iff \neg(\mathbf{o}_{tm} \wedge \mathbf{o}_{t'm}), \quad m \in \mathcal{M}, a, a' \in \mathcal{A}^{\mathrm{m}}_m, \tag{1f}$$

$$\mathbf{nextBlast}_{am} \in \{s_b | b \in B\}, \quad a \in \mathcal{A}^{(\mathrm{al})}, \tag{1g}$$

$$\mathbf{d}^{(\mathrm{al})}_a = \begin{cases} d^{(\mathrm{al})} & \text{if } s_{am} + D_a + d^{(\mathrm{al})} < \mathbf{nextBlast}_{am} \\ d^{(\mathrm{al})} - d^{(\mathrm{b})} & \text{if } s_{am} + D_a + d^{(\mathrm{al})} > \mathbf{nextBlast}_{am} + d^{(\mathrm{b})} \\ d^{(\mathrm{al})} - \Delta d^{(\mathrm{al})} & \text{otherwise}, \quad a \in \mathcal{A}^{(\mathrm{al})}, \end{cases} \tag{1h}$$

$$s_{am} + D_a + \mathbf{d}^{(\mathrm{al})}_a \leq s_{(a+1)m}, \quad f \in \mathcal{F}, a \in \mathcal{A}^{\mathrm{f}}_f \cap \mathcal{A}^{(\mathrm{al})}, m \in \mathcal{M}_a, \tag{1i}$$

$$s_{am} + D_a \leq s_{(a+1)m}, \quad f \in \mathcal{F}, a \in \mathcal{A}^{\mathrm{f}}_f \setminus \mathcal{A}^{(\mathrm{al})}, m \in \mathcal{M}_a, \tag{1j}$$

where $\Delta d^{(\mathrm{al})} = s_{am} + D_a + d^{(\mathrm{al})} - \mathbf{nextBlast}_{am}$.

2.2 Priority-based heuristic

The main principle of the heuristic is to iteratively extend and solve the problem with more task instances and to fix a majority – but not all – of the decisions made in each iteration. To define the iterations of the heuristic, we introduce a batching of the tasks as illustrated in Figure 1. This batching is done such that there is a reasonable amount of decisions to be made for each batch and so that these decisions can be made in isolation for the current batch, even if they of course have dependencies to other decisions. Note especially that, for this reason, a batch always ends with an uninterruptible task.

In the first iteration of the heuristic, two batches are scheduled and thereafter, one batch at a time is added to the problem – and each such addition

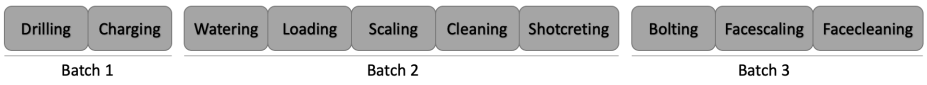

Fig. 1. The batching of tasks within a cycle

corresponds to one iteration of the heuristic. The order in which to add the batches is determined by the excavation cycle number, the priority of the face, and the batch number. Among these, the cycle number always has precedence, which means that all batches on all faces for a certain cycle number are scheduled before a next cycle is considered. For each cycle number, the order is primarily determined by the face priority, and for each face, its batches are added one at a time in accordance with the batch number.

A more formal description of the heuristic is presented through pseudo-code in Algorithm 1. Additional notation used in this description is that we let $\mathcal{F}^{(\text{prio})}$ be an ordered set of face indices, given in the order of decreasing face priority, and that we let b_i contain the tasks in batch $i \in \{1, 2, 3\}$. The scheduling within each iteration is done either by applying a CP solver to the model introduced in Section 2.1 (but adapted to batch-wise scheduling) or by using the LBBD scheme to be introduced in Section 3. In Algorithm 1, such scheduling is referred to as applying a `scheduling_method`.

Algorithm 1 Priority-based heuristic

Input: J; $\mathcal{J}_f$, $f \in \mathcal{F}$; $\mathcal{F}^{(\text{prio})}$; b_1, b_2, b_3; `scheduling_method`
for $j = 1, \ldots, J$ **do**
  $k = 1$
  while $k \leq |\mathcal{F}^{(\text{prio})}|$ **do**
    if $k = 1$ and $j = 1$ **then**
      for $i \in \{1, 2, 3\}$ **do**
        add batch b_i of cycle $j = 1$ at faces $f_1, f_2 \in \mathcal{F}^{(\text{prio})}$
        apply `scheduling_method`
      $k = k + 2$
    else
      for $i \in \{1, 2, 3\}$ **do**
        add batch b_i of cycle $j \in \mathcal{J}_f$ at face $f \in \mathcal{F}^{(\text{prio})}$
        apply `scheduling_method`
      $k = k + 1$

In detail, a batch is scheduled by applying a `scheduling_method` on a current partial problem, which includes all of the previously scheduled batches as well as the current batch. The `scheduling_method` takes the scheduling decisions for the current batch, but the previously scheduled batches are included since the current batch has to be scheduled in relation to the already scheduled

batches, e.g to be able to respect travel times. Once the `scheduling_method` has been applied, some scheduling decisions for the current batch will be fixed. When applying the LBBD scheme, these scheduling decisions are the machine assignments and the work shift assignments for each task instance. When applying a CP solver directly, these decisions are the machine assignments and the start times of each task instance.

3 Logic-based Benders decomposition

This section introduces an LBBD scheme designed to be used as a `scheduling_method` in the priority-based heuristic presented in Section 2.2. In the monolithic CP model, the objective is to minimise the sum of makespans of the faces, with the makespan of a face defined as the time when its last task instance is completed. In dialogue with Boliden, we concluded that this commonly used scheduling objective might not best reflect the decision problem to be solved in practice. Instead, we decided to focus on which shifts to schedule the task instances in, choosing as early shifts as possible, and then minimise the total duration of the task instances scheduled in the last shift used for a face. An important benefit of using this objective is that it allows for designing an LBBD scheme where the subproblem is a feasibility problem.

The decomposition is made such that in a MIP master problem, task instances are assigned to work shifts and machines are assigned to task instances. A CP subproblem is then used to schedule the tasks within the work shifts, respecting the machine assignments. Both problems are formulated in compressed time as introduced in [15]. The LBBD scheme iterates between solving the master problem and the subproblem, and if the subproblem is infeasible, a no-good cut or a set of no-good cuts are fed back to the master problem. These LBBD iterations are continued until the subproblem becomes feasible and then the master problem decisions for the current batch are fixed and returned to the heuristic. The values of the variables in the subproblem are never fixed since it is profitable if these can be adjusted depending on future master problem decisions. To enhance the performance of the LBBD scheme, cut-strengthening and problem-specific cuts are introduced, and the master problem is formulated to include a subproblem relaxation.

3.1 Master problem

The role of the master problem is to assign task instances to work shifts and machines to task instances. To formulate the MIP model, the following additional notation is used. Let the set $\mathcal{W}$ index all work shifts and let the set $\mathcal{W}^{\mathrm{l}}$ index all work shifts except the first one. Also, let the set $\mathcal{M}_t \subseteq \mathcal{M}$ index the subset of machines that can perform task $t \in \mathcal{T}$.

To handle the batches, let the set $\mathcal{F}^{(\mathrm{ba})} \subseteq \mathcal{F}$ index the currently and previously scheduled faces, and let the set $\mathcal{J}_f^{(\mathrm{ba})} \subseteq \mathcal{J}_f$ index the currently and

previously scheduled cycles at face $f \in \mathcal{F}^{(\mathrm{ba})}$. Also, let the set $\mathcal{T}_{fj} \subseteq \mathcal{T}$ index the tasks in cycle $j \in \mathcal{J}_f^{(\mathrm{ba})}$ at face $f \in \mathcal{F}^{(\mathrm{ba})}$ that are included in the currently and previously scheduled batches and denote the last included task in each $\mathcal{T}_{fj}$ by $\bar{t}_{fj}$. The main decision variables of the master problem are, for $t \in \mathcal{T}_{fj}$, $f \in \mathcal{F}^{(\mathrm{ba})}$, $j \in \mathcal{J}_f^{(\mathrm{ba})}$, $w \in \mathcal{W}$ and $m \in \mathcal{M}_t$,

$$x_{tfjwm} = \begin{cases} 1 & \text{if task } t \text{ at face } f \text{ in cycle } j \text{ is assigned to work shift } w \text{ and} \\ & \text{performed by machine } m, \\ 0 & \text{otherwise.} \end{cases}$$

The duration of a task $t \in \mathcal{T}$ is D_t and each work shift $w \in \mathcal{W}$ has a length $D_w = e_w - s_w$, where e_w and s_w are the end and start times of the work shift, respectively. The second longest duration of all tasks is denoted p and used as an aid when fitting task instances into work shifts. Some constraints are specific to batch 2 and the parameter B_{fj} is used to indicate if batch 2 has been or is currently being scheduled in cycle $j \in \mathcal{J}_f^{(\mathrm{ba})}$ at face $f \in \mathcal{F}^{(\mathrm{ba})}$.

For $f \in \mathcal{F}^{(\mathrm{ba})}$, $j \in \mathcal{J}_f^{(\mathrm{ba})}$, $w \in \mathcal{W}$ and $m \in \mathcal{M}_6$ and given that $B_{fj} = 1$ the following variables are defined to account for afterlag in the master problem,

$$y_{fjwm} = \begin{cases} 1 & \text{if task 6 and task 7 are assigned to the same work shift,} \\ 0 & \text{otherwise.} \end{cases}$$

The face and cycle last added to $\mathcal{F}^{(\mathrm{ba})}$ and $\mathcal{J}_f^{(\mathrm{ba})}$, respectively, are denoted by $\bar{f}$ and $\bar{j}_{\bar{f}}$. The objective value is represented by the auxiliary variable $o_{\bar{f}}$ which is defined by constraint (2n) below. The master problem is

$$\begin{aligned}
&\min o_{\bar{f}}, \\
&\text{s.t.} \sum_{w \in \mathcal{W}, m \in \mathcal{M}_t} x_{tfjwm} = 1,\ t \in \mathcal{T}_{fj}, f \in \mathcal{F}^{(\mathrm{ba})}, j \in \mathcal{J}_f^{(\mathrm{ba})}, && \text{(2a)} \\
&\sum_{w \in W, m \in \mathcal{M}_{t-1}} s_w x_{(t-1)fjwm} \le \sum_{w \in W, m \in \mathcal{M}_t} s_w x_{tfjwm},\ t \in \mathcal{T}_{fj}^{\mathrm{l}}, f \in \mathcal{F}^{(\mathrm{ba})}, j \in \mathcal{J}_f^{(\mathrm{ba})}, && \text{(2b)} \\
&\sum_{w \in W, m \in \mathcal{M}_{t-1}} e_w x_{(t-1)fjwm} \le \sum_{w \in W, m \in \mathcal{M}_t} s_w x_{tfjwm},\ t \in \tilde{\mathcal{T}}_{fj}^{\mathrm{c}}, f \in \mathcal{F}^{(\mathrm{ba})}, j \in \mathcal{J}_f^{(\mathrm{ba})}, && \text{(2c)} \\
&\sum_{w \in W, m \in \mathcal{M}_9} s_w x_{9f(j-1)wm} \le \sum_{w \in W, m \in \mathcal{M}_0} s_w x_{0fjwm},\ f \in \mathcal{F}^{(\mathrm{ba})}, j \in \mathcal{J}_f^{(\mathrm{ba}),\mathrm{l}}, && \text{(2d)} \\
&\sum_{m \in \mathcal{M}_t, t \in \mathcal{T}_{fj}^{\mathrm{c}}} D_t x_{tfjwm} + \sum_{m \in \mathcal{M}_1} (D_1 + p) x_{1fjwm} + \\
&\quad B_{fj}\Big(\sum_{m \in \mathcal{M}_7} p x_{7fjwm} + \sum_{m \in \mathcal{M}_6} d^{(\mathrm{al})} y_{fjwm}\Big) \le D_w + p, \\
&\quad f \in \mathcal{F}^{(\mathrm{ba})}, j \in \mathcal{J}_f^{(\mathrm{ba})}, w \in \mathcal{W}, && \text{(2e)} \\
&\sum_{m \in \mathcal{M}_t, t \in \mathcal{T}_{fj}} D_t x_{tfj(w-1)m} + \sum_{m \in \mathcal{M}_t, t \in \mathcal{T}_{fj}^{\mathrm{c}}} D_t x_{tfjwm} + \sum_{m \in \mathcal{M}_1} (D_1 + p) x_{1fjwm} +
\end{aligned}$$

$$B_{fj}\Big(\sum_{m\in\mathcal{M}_6} d^{(\mathrm{al})}\left(y_{fj(w-1)m}+y_{fjwm}\right)+\sum_{m\in\mathcal{M}_7} p x_{7fjwm}\Big)\leq$$
$$e_w - s_{(w-1)} + p,\ f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}^{\mathrm{l}}, \tag{2f}$$
$$B_{fj}\sum_{t\in\mathcal{T}_s, m\in\mathcal{M}_t}(x_{tfj(w-1)m}+x_{tfjwm})\leq 3,\ w\in\mathcal{W}, f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, \tag{2g}$$
$$B_{fj}\sum_{m\in\mathcal{M}_7}(x_{7fjwm}+x_{6fjw\tilde{m}}-y_{fjw\tilde{m}}-1)\leq 0,$$
$$f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}, \tilde{m}\in\mathcal{M}_6, \tag{2h}$$
$$B_{fj}(y_{fjwm}-x_{6fjwm})\leq 0,\ f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}, m\in\mathcal{M}_6, \tag{2i}$$

$$B_{fj}\sum_{m\in\mathcal{M}_7}(x_{7fjwm}-y_{fjw\tilde{m}})\geq 0,$$
$$f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}, \tilde{m}\in\mathcal{M}_6, \tag{2j}$$
$$\sum_{j\in\mathcal{J}_f^{(\mathrm{ba})}, f\in\mathcal{F}^{(\mathrm{ba})}} x_{7fjwm}\leq 1,\ w\in\mathcal{W}, m\in\mathcal{M}_7, \tag{2k}$$
$$\text{[no-good cuts]}, \tag{2l}$$
$$\text{[problem-specific cuts]}, \tag{2m}$$
$$\sum_{m\in\mathcal{M}_{\bar{t}}} s_w x_{\bar{t}\bar{f}\bar{j}wm}+\sum_{t\in\mathcal{T}_{\bar{f}\bar{j}}, m\in\mathcal{M}_t} D_t x_{t\bar{f}\bar{j}wm}\leq o_{\bar{f}},\ w\in\mathcal{W}, \tag{2n}$$

where for, $f\in\mathcal{F}^{(\mathrm{ba})}$ and $j\in\mathcal{J}_f^{(\mathrm{ba})}$, $\mathcal{T}_{fj}^{\mathrm{c}}=\mathcal{T}_{fj}\setminus\{2,8\}$, $\mathcal{T}_{fj}^{\mathrm{l}}=\mathcal{T}_{fj}\setminus\{0\}$ and $\tilde{\mathcal{T}}_{fj}^{\mathrm{c}}=\mathcal{T}_{fj}\cap\{2,8\}$. Additionally, $\bar{t}$, $\bar{f}$, and $\bar{j}$ denote the last task, face, and cycle, respectively, in the batches currently considered, and $\mathcal{J}_f^{(\mathrm{ba}),\mathrm{l}}$ is the set of cycles in $\mathcal{J}_f^{(\mathrm{ba})}$ excluding the first cycle and $\mathcal{T}_s=\{6,7,8,9\}\subset\mathcal{T}$.

The objective essentially directs the master problem to schedule each task instance in the earliest possible work shift, and to leave as short total task duration as possible for the last used shift. Constraint (2a) assigns exactly one machine to each task instance. Constraints (2b) and (2d) make sure that consecutive task instances are assigned to the same or to consecutive work shifts, while constraint (2c) prevents the pairs *charging* and *watering*, and *bolting* and *facescaling*, respectively, to be assigned to the same work shifts since it is impossible for those pairs of task instances to be scheduled in the same shift. Constraints (2e), (2f) and (2g) restrict which task instances that can be placed in the same work shifts, essentially removing some assignments that will be infeasible in the subproblem. Constraints (2h)–(2j) define the value of the afterlag variable. Constraint (2k) bounds the number of task instances a bolting machine can be assigned to in a work shift. Finally, the no-good cuts and the problem-specific cuts (to be introduced in Section 3.3) generated so far in the solution process are represented

by (2l) and (2m), respectively. Note that constraints (2c) and (2h)–(2k) form a subproblem relaxation that strengthens the master problem.

3.2 Subproblem

A CP solver is applied to solve the subproblem, using a model derived from the monolithic model in Section 1. The adjustments made to account for the master problem decisions are as follows. Let the set $\mathcal{A}^{\text{curr}}$ index the task instances that are to be scheduled by the subproblem and let $\bar{w}_a$ denote the work shift assigned to task instance $a \in \mathcal{A}^{\text{curr}}$. The machine assigned to task instance $a \in \mathcal{A}^{\text{curr}}$ is denoted by $\bar{m}_a$ and the set that includes the indices of the used machines is denoted by $\bar{\mathcal{M}}$. Due to the fix machine assignments, the interval variables are no longer optional and constraint (1a) is not needed in the subproblem model. Furthermore, the domains of the interval variables will be restricted to their assigned work shifts by the constraints (3a) and (3b) and for this reason, constraint (1c) is not needed in the subproblem. Moreover, the scheduling of the blasting tasks does not need to be handled in the subproblem, since the master problem assignment of the charging and watering tasks imply when blasting will occur. Hence, constraint (1b) does not need to be included in the subproblem and the **nextBlast** variables are fixed, resulting in that also constraint (1g) is redundant to include in the subproblem model. Lastly, the subproblem has no objective function since the objective depends only on master problem decisions. Hence, the constraints to be included in the subproblem model are

$$\mathbf{startOf}(I_{a\bar{m}_a}) \in \{s_{\bar{w}_a}, \dots, e_{\bar{w}_a} - D_a\},\ a \in \mathcal{A}^{(\text{un})} \cap \mathcal{A}^{\text{curr}}, \tag{3a}$$

$$\mathbf{startOf}(I_{a\bar{m}_a}) \in \{s_{\bar{w}_a}, \dots, e_{\bar{w}_a} + D_a\},\ a \notin \mathcal{A}^{(\text{un})} \cap \mathcal{A}^{\text{curr}}, \tag{3b}$$

$$\mathbf{b}_{aa'} = 1 \iff \mathbf{s}_{a\bar{m}_a} + D_a + l_{F_a F_{a'}} < \mathbf{s}_{a'\bar{m}_a},\ m \in \bar{\mathcal{M}},\ a, a' \in \mathcal{A}^{\text{m}}_m \cap \mathcal{A}^{\text{curr}}, \tag{3c}$$

$$\mathbf{b}_{aa'} = 0 \iff \mathbf{s}_{a'\bar{m}_{a'}} + D_{a'} + l_{F_{a'} F_a} < \mathbf{s}_{a\bar{m}_a},\ m \in \bar{\mathcal{M}},\ a, a' \in \mathcal{A}^{\text{m}}_m \cap \mathcal{A}^{\text{curr}}, \tag{3d}$$

$$\mathbf{d}^{(\text{al})}_a = \begin{cases} d^{(\text{al})} & \text{if } \mathbf{s}_{a\bar{m}_a} + D_a + d^{(\text{al})} < \mathbf{nextBlast}_{a\bar{m}_a} \\ d^{(\text{al})} - d^{(\text{b})} & \text{if } \mathbf{s}_{a\bar{m}_a} + D_a + d^{(\text{al})} > \mathbf{nextBlast}_{a\bar{m}_a} + d^{(\text{b})} \\ d^{(\text{al})} - \Delta d^{(\text{al})} & \text{otherwise}, \quad a \in \mathcal{A}^{(\text{al})} \cap \mathcal{A}^{\text{curr}}, \end{cases} \tag{3e}$$

$$\mathbf{s}_{a\bar{m}_a} + D_a + \mathbf{d}^{(\text{al})}_a \le \mathbf{s}_{(a+1)\bar{m}_{a+1}},\ f \in \mathcal{F}, a \in \mathcal{A}^{\text{f}}_f \cap \mathcal{A}^{\text{curr}} \cap \mathcal{A}^{(\text{al})}, \tag{3f}$$

$$\mathbf{s}_{a\bar{m}_a} + D_a \le \mathbf{s}_{(a+1)\bar{m}_{a+1}},\ f \in \mathcal{F}, a \in \mathcal{A}^{\text{f}}_f \cap \mathcal{A}^{\text{curr}} \setminus \mathcal{A}^{(\text{al})}, \tag{3g}$$

where $\Delta d^{(\text{al})} = \mathbf{s}_{am} + D_a + d^{(\text{al})} - \mathbf{nextBlast}_{am}$.

3.3 Feasibility cuts

Whenever the subproblem is infeasible, that information is fed back to the master problem in form of a no-good cut [3, 4] on the form

$$1 - \sum_{a \in \bar{A}} x_{a\bar{w}_a\bar{m}_a} \ge 1,$$

using the mapping $a = (tfj)$ in the definition of the variable x and the set $\bar{\mathcal{A}}$ to index the task instances included in the cut. For a no-good cut that originates from solving the subproblem, $\bar{\mathcal{A}} = \mathcal{A}^{\text{curr}}$ holds. Since such cut is not very strong, we designed and applied the following two cut-strengthening methods.

The first cut-strengthening method, presented in Algorithm 2, is shift-based and applied in the first iteration of the heuristic when the subproblem is still of small size. It strengthens a no-good cut through three steps: (I) Start at work shift 0 and add one work shift at a time to the subproblem, until it becomes infeasible. (II) Apply a *greedy cut strengthening algorithm* [4] with respect to the added work shifts, starting at shift 0. (III) Apply a *deletion filter* [3, 4] on the task instances assigned to the remaining shifts. This method gives an irreducible cut since, in the last step, a deletion filter is applied to a subset of task instances that yielded an infeasible subproblem.

Algorithm 2 Shift-based method

```
let w = 0
while subproblem feasible do
    add all task instances assigned to work shift w to the subproblem
    solve subproblem
    let w = w + 1
let w = 0
while subproblem infeasible do
    remove all task instances assigned to work shift w from the subproblem
    solve subproblem
    let w = w + 1
apply a deletion filter on the remaining task instances
```

The cuts obtained by applying Algorithm 2 are also used to derive additional no-good cuts by creating permutations of the machine assignments, using the fact that the machines are identical. For example, if one of the drillrigs cannot execute two tasks instances, then neither can any of the other two drillrigs.

The second cut-strengthening method, presented in Algorithm 3, is applied in all but the first iteration of the heuristic. This method is batch-based and designed to efficiently handle the addition of new batches as they are added by the heuristic. It strengthens a cut by applying a deletion filter on the task instances of the current batch. Assignments from previous batches are already proven to be feasible before the addition of the current batch, but because of dependencies between the batches, the current one cannot be evaluated in isolation. However, to sustain a moderate size of the subproblems to be solved during cut strengthening also in later iterations of the heuristic, only a few work shifts and their task instances from previous batches are included. The risk of this is that the strengthening can fail and the original cut needs to be used, but in practice, this never became the case.

Algorithm 3 Batch-based method

1: let $\mathcal{B}^{(\text{curr})}$ contain the task instances in the current batch
2: let $\tilde{w}$ be the earliest work shift of a task instance in $\mathcal{B}^{(\text{curr})}$
3: add all task instances in work shifts $\tilde{w}-4,\ldots,\tilde{w}$ to the subproblem
4: add all task instances $\mathcal{B}^{(\text{curr})}$ to the subproblem
5: **if** subproblem infeasible **then**
6: apply a deletion filter with respect to the task instances in $\mathcal{B}^{(\text{curr})}$

Problem specific cuts During the computational experiments, it was discovered that the master problem was particularly weak with respect to one aspect of the assignments and that many no-good cuts were required to reach feasibility due to this. The aspect originates from the limited number of machines to use for the tasks in batch 2. Specifically, if all task instances of two batches of batch 2 are scheduled within the same work shift, it is not possible to schedule all task instances of a next batch 2 within this shift.

During the solution process, it is possible to keep track of the work shifts that have been assigned two complete batches of batch 2. Let the set $\bar{\mathcal{W}}$ index such work shifts and let $\tilde{f}$ and $\tilde{j}$ be the face and cycle of the second batch that is currently scheduled. The problem-specific cut

$$\sum_{t\in b_2}\sum_{m\in\mathcal{M}_t} x_{t\tilde{f}\tilde{j}wm} \leq 4, \qquad w\in\bar{\mathcal{W}},$$

ensures that, for each work shift with at least two batches of batch 2 already scheduled, at least one task instance from the currently scheduled batch 2 is assigned to another work shift.

4 Computational results

This section presents computational results from the implementation and evaluation of the following three solution methods.

- **M-CP:** Apply a CP solver directly on a monolithic model from previous work. The model is described in Section 2.1.
- **H-CP:** Apply the priority-based heuristic and, in each iteration, use a CP solver on an adaptation of the model in Section 2.1.
- **H-LBBD:** Apply the priority-based heuristic and, in each iteration, use the LBBD scheme introduced in Section 3.

The evaluation measures that we use are computational times and the sum of the makespans for the individual faces, henceforth simply referred to as makespan. Important to note is that the only method that actually uses makespan as the sole objective is M-CP, while both H-CP and H-LBBD include a priority order, not as part of the mathematical model but through the heuristic. In addition, H-LBBD uses a slightly changed objective function as part of the LBBD

scheme design. This means that for H-LBBD, the makespan is computed after the solution is returned. In our comparison, M-CP is considered as the benchmark method, both because the method is exact and because the objective of the model is the same as our measure for evaluating the quality of the schedule. Furthermore, since M-CP stems from a re-implementation of a model in previous work, it is also the best comparison we could make in this respect.

For the evaluations, we used 6 industrially relevant instances constructed in dialogue with Boliden. These have been made publicly available and are described in more detail in Section 4.1 together with a description of computational settings. To evaluate the impact of the priority-based heuristic, each instance was solved for three different priority orders, denoted P1, P2, and P3. In P1, there is an increasing priority from the first to the last face, and for the other two, there is a randomly chosen order.

The first set of computational results, presented in Section 4.2, provides a comparison between H-CP and H-LBBD to evaluate the impact of applying the LBBD scheme instead of a CP solver in each iteration of the heuristic. This comparison is direct in terms of only evaluating the use of the LBBD scheme. Since the results of H-LBBD are much stronger than those of H-CP, the latter is omitted from further comparisons. The second set of computational results gives a comparison between H-LBBD and M-CP and is presented in Section 4.3.

4.1 Instances and computational settings

All our instances are constructed for a given mine topology that defines the distances between the different faces and for a given machine park. Each instance is characterised by the number of parallel excavation faces and the number of excavation cycles at each face. They have been encoded with #**F**:#**C**, e.g an instance with 6 parallel faces and 4 cycles at each face is encoded **6F:4C**, in line with notation used previous work [11]. Most of our instances consist of 24 cycles in total, since this gives a realistic instance size according to Boliden. An instance with a specific characteristic is **18F:XC** that has a total of 24 cycles, where faces 1 to 12 have one cycle and 13 to 18 have two cycles. Since the choice of the starting task at each face has a large impact on the objective value, we use the same starting task, drilling, for all faces and all instances.

The number of task instances that a problem instance contains is #**F** $\times$ #**C** $\times$ 10. For example, **6F:4C** include $6 \times 4 \times 10 = 240$ task instances. The machine park used in our instances consists of seven different kinds of machines, 2 *drillrigs*, 2 *chargers*, 2 *watering trucks*, 5 *loaders*, 3 *scalers*, 2 *shotcreters* and 4 *bolters*. Note that the choice of machine for each task instance also yields additional decisions to be made as part of the scheduling.

The CP models were solved by IBM ILOG CP Optimizer version 20.1.0.0 and the MIP model was solved using Gurobi Optimizer version 9.1.2. All tests were run on a PC using an Intel i7 2600k processor at 4.1 GHz and 16 GB of RAM. The industrially relevant instances have been generated in dialogue with Boliden and made publicly available at https://gitlab.liu.se/eliro15/underground_mining_instances where further details are found.

4.2 Comparison between H-LBBD and H-CP

The first set of experiments is made to determine which `scheduling_method` yields the best performance of the heuristic. Since the only difference between the methods H-CP and H-LBBD is how the scheduling is done in each iteration of the heuristic, it is possible to make an individual comparison for each pair of instance and priority order, providing 18 such pairs to evaluate.

Initial tests revealed that it would not be possible to let the CP solver run to optimality in each iteration of the heuristic, and for this reason we needed to put a time limit on each iteration. Since the initial tests also indicated that the LBBD scheme was consistent in producing reasonable schedules in each iteration, without time limits and with a zero-valued MIP-gap in the master problem, we decided to record the time used by H-LBBD in each iteration of the heuristic and then give the CP solver the same amount of time in each iteration. The results from the comparisons with these time limits are shown in Table 1.

Instance -priority	Time [s]	H-CP	H-LBBD
6F:4C -P1	214	-	**71191**
-P2	418	79718	**72156**
-P3	221	-	**71191**
10F:3C -P1	1005	114291	**98046**
-P2	1281	112792	**96375**
-P3	1646	98595	**97683**
12F:2C -P1	1237	95777	**83321**
-P2	1390	91255	**85279**
-P3	835	93228	**82722**
F15:C2 -P1	2295	141988	**116935**
-P2	2225	**118584**	119523
-P3	2651	128531	**119784**
18F:XC -P1	1159	99879	**98917**
-P2	1034	103966	**99829**
-P3	1092	105670	**104111**
24F:1C -P1	2819	134634	**126304**
-P2	2017	132404	**125704**
-P3	2767	131204	**127434**

Table 1. The makespan (or "-" when no solution was found) for the schedules produced by H-CP and H-LBBD, respectively, when using the same total computational time for an instance-priority pair

The results in Table 1 show that H-LBBD yields a better makespan than H-CP for all instances except **15F:2C** with priority P2. In particular, H-LBBD excels when the ratio of cycles to faces is high. Furthermore, when comparing the results for different priority orders for the same instance, it can be noted that

the priority order has an impact on the makespan. Sometimes the differences are significant, especially for H-CP. For H-LBBD the results are more consistent. The impact of the priority order is further discussed in Section 4.3 when benchmarking with M-CP. Our conclusion from the first set of experiments is that H-LBBD performs better than H-CP and that we therefore can omit the latter in the further evaluations.

4.3 Comparison between H-LBBD and M-CP

The comparison between H-LBBD and M-CP is multifaceted since, even if in both cases we are interested in finding a solution with a short makespan quickly, both the methods and the objectives differ. Objective-wise, H-LBBD takes the priority order into account, while M-CP does not. However, in the comparison, the result is evaluated with respect to the makespan, that is, the objective used in M-CP. This means that for one result from M-CP, we have three results from H-LBBD to compare with and this comparison needs to include an analysis of the impact of taking the priority order into account in H-LBBD.

Method-wise, H-LBBD is a heuristic while M-CP is an exact method. However, as will be apparent in the results, the CP solver is far from capable of finding an optimal solution, and for this reason it becomes more reasonable to treat also M-CP as a heuristic in the sense that it is evaluated by the makespan produced after a certain time limit. For this purpose, we introduce two time limits. The first is *LBBD max time limit*, which is the maximum time spent by H-LBBD for the priority order that required the longest computational time, hence giving an advantage to M-CP in the evaluation. The second one, *Benchmark time limit*, is chosen to be significantly longer than the first to, if possible, provide a really short makespan to use as benchmark when assessing the impact of the priority order. The results are displayed in Table 2.

We begin by comparing H-LBBD with M-CP for the LBBD max time limit. Most striking is that for all instances, the makespans of the schedules from H-LBBD are, irrespective of the priority order used, shorter than the makespan of the corresponding schedule produced by M-CP. We interpret this as an effect of H-LLBD being a much more efficient solution method for the problem, and this to a magnitude that makes the priority order irrelevant. Comparing the relative improvement of the makespan between M-CP and H-LBBD for the priority order with the best makespan gives a relative difference of 22%, 6%, 2%, 14%, and 2%, respectively, for the instances solved by both methods.

When comparing the makespans for the LBBD max time limit and the Benchmark time limit for M-CP, we note that there is an improvement when allowing the CP solver additional computational time. For **6F:4C** it makes the difference between finding a feasible solution and not, and in this case the time limit was increased by a factor of about 9. For the other instances, the relative improvements of the makespans are 5.3%, 1.8%, 0.04%, 6.7%, and 0.1%, respectively, for the time limit being increased by a factor of 3, 3, 3, 5, and 3, respectively.

The final comparison is between H-LBBD and M-CP with the Benchmark time limit. There we see that for instance **15F:2C** with priority P2 and P3, the

Instance	Eval. measure	Method				
		M-CP		H-LBBD		
		Benchmark time limit	LBBD max time limit	P1	P2	P3
6F:4C	Makespan	83004	-	**71191**	72156	**71191**
	Time [s]	3600	418	214	418	221
10F:3C	Makespan	117249	123795	98046	**96375**	97683
	Time [s]	5400	1646	1005	1281	1646
12F:2C	Makespan	86577	88200	83321	85279	**82722**
	Time [s]	5400	1390	1237	1390	835
15F:2C	Makespan	119351	119863	**116935**	119523	119784
	Time [s]	7200	2651	2295	2225	2651
18F:XC	Makespan	107524	115294	**98917**	99829	104111
	Time [s]	5400	1159	1159	1034	1092
24F:1C	Makespan	126924	128273	126304	**125704**	127434
	Time [s]	7200	2819	2819	2017	2767

Table 2. The makespan (or "-" when no solution was found) for the schedules produced by M-CP and H-LBBD, respectively, for different time limits

makespan is longer in the schedules from H-LBBD than in the schedule from M-CP. However, the relative differences are only about 0.1% and 0.3%, respectively, after about 3 times more computational time. For all other instances and priority orders, H-LBBD still provides the best makespans – despite the much longer runtimes for M-CP. Again, this means that the efficiency of the LBBD scheme dominates that of the effect of taking the priority order into account. Further analysis of the impact of priority orders is therefore left for future work. As a preliminary result in this direction, Figure 2 illustrates the schedules from H-LBBD for instance **10F:3C** with priority order P1 and P2, respectively. More illustrations of this kind are found in [6].

5 Concluding remarks

This paper addresses how to formulate and solve a short-term scheduling problem for a cut-and-fill mine. To enable a high degree of control on the scheduling output, a priority-based heuristic that adapts to different face priorities was proposed. Note that since the distances between different faces of a mine are not the same, the priority order is expected to have an impact on the best possible makespan that can be obtained. The heuristic was integrated with an LDDB scheme to solve the partial scheduling problems in each iteration of the heuristic. To enable an efficient LBBD scheme, a new objective for the problem was introduced. This objective indirectly aims for a short makespan but it uses the shift structure of the problem instead of considering the end times of tasks.

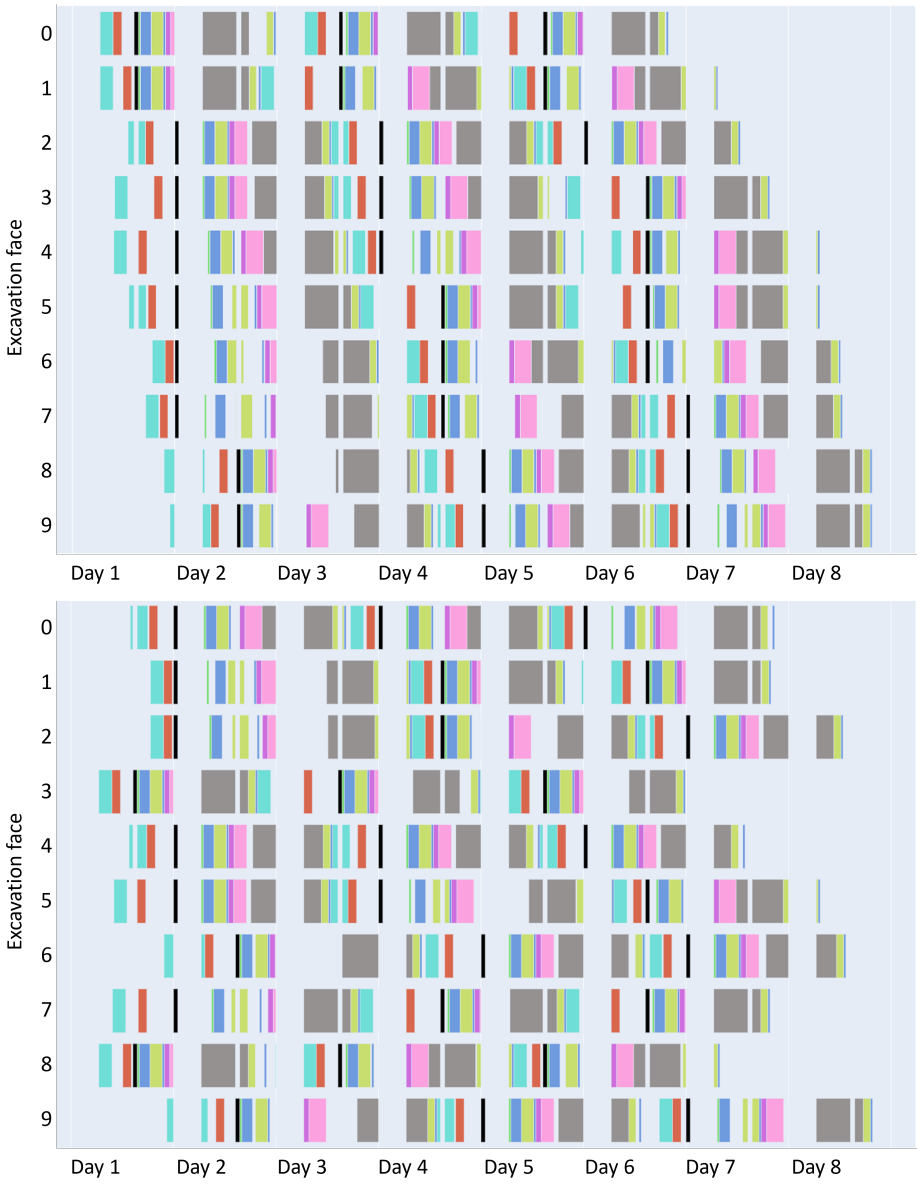

Fig. 2. Schedules for instance **10F:3C** for the priority order P1 (order: 1,2,3,4,5,6,7,8,9,10) and P2 (order: 4,9,6,5,1,8,3,2,7,10), respectively. Colours are: turquoise for drillrigs 1–2, red for chargers 1–2, green for watering trucks 1–2, blue for loaders 1–5, lime green for scalers 1–3, purple for shotcreters 1–2, grey for bolters 1–4, and pink for afterlag

The effect of including a priority order was evaluated by comparing with the makespans of schedules generated without considering a priority. The results showed that – thanks to computational efficiency – our schedules had a shorter makespan than that obtained when applying a CP solver on a monolithic CP model aiming only at minimising the makespan. Note that this was almost always the case also when the CP solver was given significantly longer computational times. The conclusion we draw from this is that applying an approach like the one we propose shows great promise for this type of scheduling problem and that the impact of including priorities needs to be further studied.

In other future work, it is relevant both to improve the modelling of the problem and to continue the development of more efficient solution methods. In particular, the heuristic can be further developed and so can the LBBD scheme. For the latter, there is potential to improve the master problem formulation and the cut-strengthening procedures.

References

1. Coban, E., Hooker, J.: Single-facility scheduling by logic-based Benders decomposition. Annals of Operations Research **210**, 245–272 (2013)
2. Emde, S., Polten, L., Gendreau, M.: Logic-based Benders decomposition for scheduling a batching machine. Computers & Operations Research **113**, 104777 (2019)
3. Hooker, J.N.: Logic-Based Benders Decomposition for Large-Scale Optimization, pp. 1–26. Springer International Publishing, Cham (2019)
4. Karlsson, E., Rönnberg, E.: Strengthening of feasibility cuts in logic-based Benders decomposition. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 45–61. Springer International Publishing, Cham (2021)
5. Karlsson, E., Rönnberg, E.: Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling. Computers & Operations Research (2022). https://doi.org/10.1016/j.cor.2022.105916
6. Lindh, E., Olsson, K.: Scheduling of an underground mine by combining logic-based Benders decomposition and a constructive heuristic. Master's thesis, Department of Mathematics, Linköping University, Sweden (2021)
7. Nehring, M., Topal, E., Knights, P.: Dynamic short term production scheduling and machine allocation in underground mining using mathematical programming. Mining Technology **119**(4), 212–220 (2010)
8. Roshanaei, V., Luong, C., Aleman, D.M., Urbach, D.: Propagating logic-based Benders' decomposition approaches for distributed operating room scheduling. European Journal of Operational Research **257**(2), 439–455 (2017)
9. Schulze, M., Rieck, J., Seifi, C., Zimmermann, J.: Machine scheduling in underground mining: an application in the potash industry. OR Spectrum **38**, 365–403 (2015)
10. Song, Z., Schunnesson, H., Rinne, M., Sturgul, J.: Intelligent scheduling for underground mobile mining equipment. PLOS ONE **10**(6), 1–21 (2015)
11. Åstrand, M.: Short-term Underground Mine Scheduling: An Industrial Application of Constraint Programming. No. 36, KTH, Automatic Control (2021), PhD thesis

12. Åstrand, M., Johansson, M., Feyzmahdavian, H.R.: Short-term scheduling of production fleets in underground mines using CP-Based LNS. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 365–382. Springer International Publishing, Cham (2021)
13. Åstrand, M., Johansson, M., Greberg, J.: Underground mine scheduling modelled as a flow shop : a review of relevant work and future challenges. The Southern African Journal of Mining and Metallurgy **118**(12), 1265–1276 (2018)
14. Åstrand, M., Johansson, M., Zanarini, A.: Fleet scheduling in underground mines using constraint programming. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 605–613. Springer International Publishing, Cham (2018)
15. Åstrand, M., Johansson, M., Zanarini, A.: Underground mine scheduling of mobile machines using constraint programming and large neighborhood search. Computers & Operations Research **123**, 105036 (2020)