Fleet Scheduling in Underground Mines using Constraint Programming

Max Åstrand [1,2,*], Mikael Johansson [1], and Alessandro Zanarini [3]

max.astrand@se.abb.com, mikaelj@kth.se, alessandro.zanarini@ch.abb.com

[1] KTH Royal Institute of Technology, Stockholm, Sweden
[2] ABB Corporate Research Center, Västerås, Sweden
[3] ABB Corporate Research Center, Baden-Dättwil, Switzerland
[*] Corresponding author

Abstract. The profitability of an underground mine is greatly affected by the scheduling of the mobile production fleet. Today, most mine operations are scheduled manually, which is a tedious and error-prone activity. In this contribution, we present and formalize the underground mine scheduling problem, and propose a CP-based model for solving it. The model is evaluated on instances generated from real data. The proposed approach shows promising results and a potential for further extensions.

1 Introduction

Mining is the process of extracting minerals from the earth, commonly done either in open pit or underground mines. The margins in underground mining are constantly under pressure since costs increase as production goes deeper. One major component influencing the operational performance, and thus the profitability, of an underground mine is how the mobile machinery is coordinated. Today, the machine fleet is often scheduled manually with methods on the brim of what they can handle and with no performance guarantees. In a survey of more than 200 high level executives from mining companies all around the world [1] maximizing production effectiveness was identified as the top challenge for modern mines, even more than improving reliability of individual equipment. This highlights the importance of system level coordination.

The operation of a mine is often planned on different levels, each with its own time horizon and task granularity. The life-of-mine plan, which contains a rough strategy for which year to extract what parts of the ore-body, has the longest time-span. Based on the life-of-mine plan, extraction plans of various granularity are constructed. These have shorter time horizons and include more details about what amount of ore is planned during precise periods of time. The extraction plans are executed by allocating machines and personnel to extraction activities and scheduling them over a horizon of typically 1 week.

To the best of the authors' knowledge, there is no previous work on using CP to schedule the mobile production fleet in underground mining. The most similar

problem can be found in [2], where they study scheduling mobile machines in an underground potash mine using MIP-models and construction procedures. Another underground mine scheduling problem is described in [3] where schedules are created by first enumerating all activity sequences and then selecting the one with the shortest makespan. To cope with the factorial growth of possible permutations the authors cluster the faces based on geographical distance. The authors in [4] study how to transport ore in an underground mine, via intermediate storages, to the mine hoist. They use a MIP-model to allocate machines to different drawpoints on a shift basis over a period of 2 months. The authors refine their work in [5] and propose several simplifications which allow to decrease computation time.

The authors in [6] study an open pit mine, where mine regions are scheduled on a weekly basis. Using a multiobjective approach, they study over 40 objectives, including maximizing the utilization of trucks and minimizing deviations from targeted production. In [7] a model is developed for scheduling numerous autonomous drill rigs in an open pit mine. The problem naturally decomposes into subproblems, and the authors develop search heuristics for each subproblem. A high-level CSP, linking the subproblems, is solved to find a solution in the joint search space.

Another underground case is introduced in [8], where they study the effect of using different dispatch strategies for routing machines to locations in a diamond mine. The authors conclude that strategies that separate the LHDs geographically seem beneficial. This is due to the confined environment, in which avoiding deadlocks (e.g. machines meeting each other in a one-way tunnel) is crucial. Another study of underground routing can be found in [9], where the authors continue the work previously found in [10].

Our contribution is the first study of production scheduling in underground mining using CP. Further, the problem introduced generalizes similar problems studied by other authors (using other methods). Most notably, we impose and exploit the presence of blast windows, allow for a mix of interruptible and non-interruptible tasks, and support tasks that have an after-lag.

The paper is organized as follows: Section 2 describes the problem and introduces the necessary notation. A CP-based model for solving the problem at hand is then developed in Section 3. Section 4 reports experimental results on instances generated by data coming from a real mine. Conclusions are drawn in Section 5.

2 Problem Description

The mine operations that we consider are located at underground sites called *faces*, which denotes the end of an underground tunnel. From here-on, the faces are labeled $F = \{1, \ldots, n\}$. In order to extract ore from the mountain, a periodic sequence of activities takes place at each face: C = *(drilling, charging, blasting, ventilating, washing, loading, scaling, cleaning, shotcreting, bolting, face scaling, face cleaning)*. We refer to a full period of activities as a *cycle*. All the activities

except blasting and ventilation require a specific (today) human-operated machine to be used, and we denote with $\hat{C}$ this subset of activity types. Specifically, drilling requires a drill rig in order to drill holes in the face processed; when charging, the holes are filled with explosives typically using a wheel loader with a platform; after blasting, the rock that has been separated from the mountain is sprayed with water to reduce the amount of airborne particles. The rock is then removed (loaded) from the face with a Load, Haul, Dump machine (LHD); smaller rocks loosely attached to the mountain are later mechanically removed with a scale rig (scaling), and removed from the drift with an LHD (cleaning); the two successive steps are for ensuring safety, namely to secure the rock to the insides of the tunnel (bolting), and spraying the insides of the tunnel with concrete (shotcreting). Finally, the face is prepared for the next cycle by a scaling rig (face scaling), and the separated rock is removed by an LHD (face cleaning). For each activity type $c \in \hat{C}$, a non-empty set of machines M_c is available in the mine to perform that specific operation. In the general case we are studying, some machines can be employed to perform different activity types.

Blasting is a key activity in underground mine operations and it sets the overall pace of production for the entire mine. It is common that blasts occur in predetermined time windows during the day (typically twice or thrice); for safety reasons, no human operator should be inside the mine during the blasting time window and subsequent ventilation of toxic blast fumes, independently of whether blasting occurs, and on which faces; in other words, no other activity can take place at any face across the whole mine.

As blasting and ventilation have the same properties, happen one after the other, and affect the scheduling in the same manner, in the following, whenever we refer to blasts we denote an activity that spans over the duration of the blast and the subsequent ventilation. We refer to the candidate blast time windows as $B = \{(s_{b_1}, e_{b_1}), (s_{b_2}, e_{b_2}), (s_{b_3}, e_{b_3}), \dots\}$ where the pair (s_{b_i}, e_{b_i}) defines the start and end of the time window; for simplicity, and without loss of generality, we assume that each time window has equal duration $d_b = e_{b_\cdot} - s_{b_\cdot}$ and that the blasting (and ventilation) can fit inside the time window (the problem would be trivially infeasible otherwise).

Most activity types are interruptible, i.e. they can start prior to a blast window, then be suspended during the blast, and resumed after the blast window. *Shotcreting* is however not interruptible; furthermore the subsequent activity (*bolting*) can only happen after a delay required for the concrete to cure; this delay, also refer to as after-lag, has a duration of d_{al}.

An instance of a problem is composed of a set of activities a_i^f indicating the i'th activity at face $f \in F$; the sequence of activities in each face is defined by $A^f = (a_1^f, \dots, a_{m_f}^f)$: it consists of an arbitrary number of extraction cycles that follow the sequence defined in C. We define by $d(a_i^f)$ the nominal duration of activity a_i^f, i.e. the duration in case no interruption takes place. Note that all the activity durations are in practice shorter than the time between two blasts, i.e. $d(a_i^f) < d_b$. Furthermore, let $c(a_i^f) \in C$ be a function indicating the activity type of activity a_i^f and let $I_\cdot(A^f) = \{i \mid c(a_i^f) = \cdot\}$ be the set of indices of the

activities at face A^f of a given type; in this way, $I_{shotcreting}(A^f)$, for example, indicates the indices of all the shotcreting activities at face A^f.

Since the schedules are deployed in a rolling-horizon approach, where each face have a predefined number of cycles to be performed during the life of the mine, it makes sense to use an objective function that accounts for the state of all faces. Therefore, the scheduling problem consist of allocating the available mining machinery to the activities, and schedule them in order to minimize the sum of the makespans of all faces.

3 Model

The problem resembles a rich variant of the flow-shop problem, with additional aspects such as unavailabilities due to blasts, after lags, and a mix of interruptible and uninterruptible activities [11, 12]. For each activity a_i^f, we employ $|M_{c(a_i^f)}|$ conditional interval variables [13] representing the potential execution of that activity on the candidate machines[1]. Specifically, each conditional interval variable consists of a tuple of four integer variables: $\mathtt{s}_{ir}^f$, $\mathtt{d}_{ir}^f$, $\mathtt{e}_{ir}^f$, $\mathtt{o}_{ir}^f$ indicating the start time, the duration, the end time ($\mathtt{s}_{ir}^f + \mathtt{d}_{ir}^f = \mathtt{e}_{ir}^f$) and the execution status of activity a_j^f on machine $r \in M_{c(a_i^f)}$, respectively. This model allows for machine-dependent duration in case machines have different throughput.

The execution status takes value 0 if the interval is not executed with machine r, or 1 if it is executed with machine r. As each activity is executed using exactly one machine:

$$\sum_{r \in M_{c(a_i^f)}} \mathtt{o}_{ir}^f = 1 \quad \forall f \in F \quad \forall i \in 1, \dots, |A^f| \tag{1}$$

The start times for blasts must be aligned with the blast time windows, therefore:

$$\mathtt{s}_{ir}^f \in \{s_{b_1}, s_{b_2}, s_{b_3}, \dots\} \quad \forall f \in F \quad \forall i \in I_{blasting}(A^f) \quad \forall r \in M_{blasting} \tag{2}$$

Uninterruptible tasks (namely shotcreting) also must not overlap with the blast time windows, independently of whether or not a blast occurs in that specific face:

$$\mathtt{s}_{ir}^f \in \mathcal{S}_{shotcreting} \quad \forall f \in F \quad \forall i \in I_{shotcreting}(A^f) \quad \forall r \in M_{shotcreting} \tag{3}$$

where $\mathcal{S}_{shotcreting} = \{0, \dots, s_{b_1} - d(a_i^f), e_{b_1}, \dots, s_{b_2} - d(a_i^f), \dots\}$.

Both blast and shotcreting activities are non-interruptible therefore their respective durations are set to the nominal activity durations:

$$\mathtt{d}_{ir}^f = d(a_i^f) \quad \forall f \in F \quad \forall i \in I_{blasting}(A^f) \quad \forall r \in M_{blasting} \tag{4}$$

$$\mathtt{d}_{ir}^f = d(a_i^f) \quad \forall f \in F \quad \forall i \in I_{shotcreting}(A^f) \quad \forall r \in M_{shotcreting} \tag{5}$$

[1] In order to simplify the notation, we assume that for blasting activities we have a single machine $r \in M_{blasting}$ with infinite capacity

Further, all activities, except blasting, cannot start during a blast windows:

$$\mathtt{s}^f_{ir} \in \mathcal{S}_c \quad \forall f \in F \quad \forall c \in \tilde{C} \quad \forall i \in I_c(A^f) \quad \forall r \in M_c \tag{6}$$

where $\mathcal{S}_c = \{0, \ldots, s_{b_1}, e_{b_1}, \ldots, s_{b_2}, \ldots\}$ and $\tilde{C} = C \setminus \{blasting\}$.

In order to model the interruptible activities, their associated intervals have variable durations. We introduce a variable $\mathtt{p}^{\mathtt{f}}_{\mathtt{ir}}$ indicating whether the interval of face f, index i, resource r has been interrupted. We use a sum of reified constraints to go over all the possible blast time windows, and verify if there is one that overlaps with the interval; if an interval starts before a blast time window and its execution goes past the start of the blast time window then $p^f_{ir} = 1$. Note that the durations of activities are such that they will never span over two blast time windows. Finally, whenever the interval gets interrupted, its duration needs to be augmented by the duration of the blast time window.

$$\mathtt{p}^f_{ir} = \sum_k (\mathtt{s}^f_{ir} < s_{b_k}) * (\mathtt{s}^f_{ir} + d(a^f_i) > s_{b_k})$$

$$\mathtt{d}^f_{ir} = d(a^f_i) + \mathtt{p}^f_{ir} * d_b \qquad \forall f \in F \quad \forall c \in \tilde{C} \quad \forall i \in I_c(A^f) \quad \forall r \in M_c \tag{7}$$

The full order of the cyclic activities is enforced by:

$$\mathtt{s}^f_{ir} + \mathtt{d}^f_{ir} < \mathtt{s}^f_{i+1r'} \quad \forall f \in F \quad \forall i \in 1, \ldots, |A^f| - 1 \quad \forall r, r' \in M_{c(a^f_i)} \tag{8}$$

and the after-lag of shotcreting as:

$$\mathtt{s}^f_{ir} + \mathtt{d}^f_{ir} + d_{al} < \mathtt{s}^f_{i+1r'} \quad \forall f \in F \quad \forall i \in I_{shotcreting}(A^f) \quad \forall r, r' \in M_{c(a^f_i)} \tag{9}$$

Unary constraints are used for all the faces and machines to enforce disjunctive execution:

$$\mathtt{unary}(\{[\mathtt{s}^f_{ir}, \mathtt{d}^f_{ir}, \mathtt{o}^f_{ir}] \mid i = 1, \ldots, |A^f|, \ r \in M_{c(a^f_i)}\}) \quad \forall f \in F \tag{10}$$

$$\mathtt{unary}(\{[\mathtt{s}^f_{ir}, \mathtt{d}^f_{ir}, \mathtt{o}^f_{ir}] \mid f = 1, \ldots, n, \ i \in \tilde{I}^f_r\}) \quad \forall c \in C \quad \forall r \in M_c \tag{11}$$

in which $\tilde{I}^f_r = \{i \mid i \in A^f \wedge r \in M_{c(a^f_i)}\}$ indicates all the indices of the activities of face f that can be performed by machine r.

Finally, the model minimizes the sum of the makespans across all faces by (m_f is the index of the last activity of face f):

$$\sum_f \sum_{r \in M_{c(a^f_{m_f})}} \mathtt{o}^f_{m_f r} * \mathtt{e}^f_{m_f r} \tag{12}$$

3.1 Search Strategy

Different generic and ad-hoc heuristics have been tested; for brevity, we present in the following what has been experimentally deemed the most effective one.

The search proceeds in two phases: the first phase is about machine allocation, the second is about scheduling the start times. For the former, tasks are ordered by their nominal durations, and for each task the least loaded machine is chosen among the compatible ones. As for the task scheduling, tasks are chosen based on an action-based heuristic and the value selection schedules the task as early as possible. Action-based heuristic learns during the search to branch on the variables that are likely to trigger the most propagation (also known as activity-based heuristic [14]).

Randomized Restarts Some problems take a prohibitive amount of time to solve by exhaustive search. By systematically restarting the search procedure, a restart-based search samples a wider part of the search space. However, in order not to end up on the same solution after each restart, some randomness needs to be included in the search heuristics.

Therefore, the machine allocation chooses a random variable to branch on with probability p, otherwise it follows the heuristic described above. Similarly, the value selection is chosen at random with probability p, otherwise the least loaded heuristic is employed. Note that action-based branching tend to work well with restart-based search, since the restarts provide a lot of information of propagation, as well as the branching adapts itself to where it currently is in the search tree.

4 Experimental Results

The model described in Section 3 is used to solve the underground mine scheduling problem for several problem sizes. All problems are solved using the search heuristics introduced in Section 3.1, and are compared to a restart-based approach using the randomized search heuristics. All problems are solved using Gecode 5.1 with 4 threads on a laptop with an i7-7500U 2.7GHz processor.

The size of a problem is determined by *i*) the number of faces, *ii*) the number of cycles, and *iii*) the number of machines of each type. In this work, we experimented with instances with 5 and 10 faces, with 1 or 2 cycles at each face, which results in a minimum of 55 tasks and a maximum of 220 tasks to be scheduled. The considered machine parks consist of 1, or 2 machines of each type, together with a non-uniform machine park inspired by data from a real underground mine. This complex machine park consists of 4 drill rigs, 3 chargers, 1 water vehicle, 6 LHDs, 4 scaling rigs, 2 shotcreters and 3 bolters. The instances are encoded as *(# faces)F(# cycles per face)C(# machines of each type)M*. As an example, *5F2C2M* thus corresponds to a problem with 5 faces where each face has 2 cycles each, and the machine park consists of 2 machines of each type. The machine classification *CM* is used to encode the complex non-homogeneous machine park. We generated 8 different problem types and 5 instances of each problem type by starting from the nominal task durations from an operational underground mine. The task durations were perturbed by varying them randomly by a factor between -25% and +25% from the real-case scenario. The search

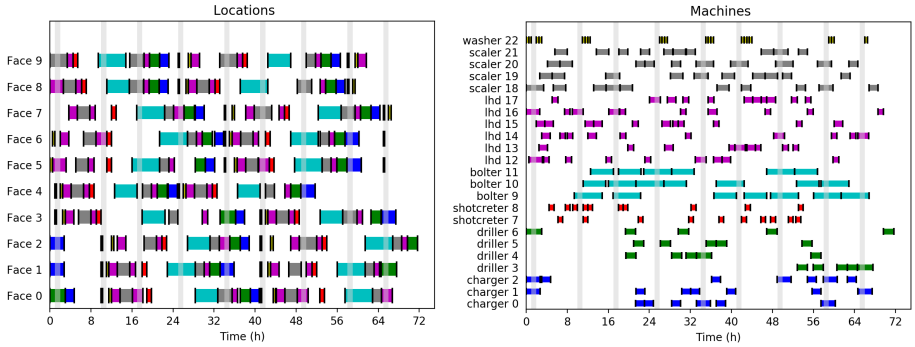

Fig. 1. The largest problem instance using 10 faces with 2 cycles at each face, together with a machine park that is based on a real underground mine.

procedure has been limited in time to 12.5 minutes for the instances with 5 faces and 25 minutes for the instances with 10 faces.

A solution to the largest problem studied, *10F2CCM*, using the restart-based search can be seen in Figure 1. The blast windows are indicated by vertically aligned grey areas. In this solution we can see some features of the problem such as shotcreting tasks are not split over blast windows (as is evident e.g. on the first shotcreting task on face 5) whereas other correctly interrupted and resumed after blast time windows. Evidently, manual scheduling quickly becomes a tedious and error-prone process when reaching real-life instance sizes.

In Table 1, we have aggregated the statistics over the 40 instances by grouping per instance type, using both the baseline and the restart-based method. The second to the fourth columns represent, respectively, the average, minimum and maximum objective across the instance set. The fifth to the seventh columns indicate, respectively, the average, minimum, maximum percentage of the objective

Table 1. Average, minimum, and maximum objective function solving 5 samples of each instance. The left column corresponds to the objective value using the baseline method, while the right column corresponds to the reduction gained by instead using the restart-based method.

	avg O_{BL}	min O_{BL}	max O_{BL}	avg $O^{\%}_{RES}$	min $O^{\%}_{RES}$	max $O^{\%}_{RES}$
5F1C1M	2082	1911	2214	0%	0%	1%
5F1C2M	1892	1751	1990	0%	0%	1%
5F2C1M	4200	3583	4575	-1%	-7%	1%
5F2C2M	4048	3560	4360	-7%	-11%	-3%
10F1C2M	4410	3973	4601	0%	-5 %	3 %
10F1CCM	3797	3606	4097	2 %	4%	1%
10F2C2M	11019	9803	12562	-15%	-11 %	-21%
10F2CCM	8222	7548	8825	-1%	-4%	2%

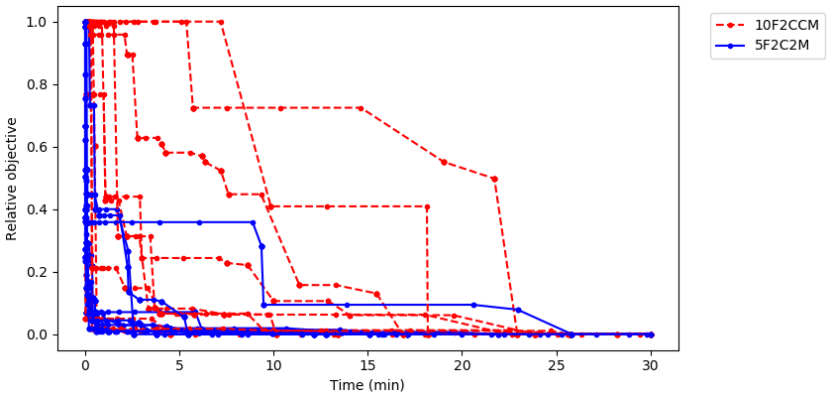

Fig. 2. How the objective value decreases over time using restart-based search. The plot shows the objective divided by the value of the first found solution for 5 samples of 10F2CCM and 5 samples of 5F2C2M.

value obtained with randomized restarts w.r.t. the deterministic baseline. It is evident that using restart-based search can be advantageous, particularly for the larger instances. Note furthermore that as shown in Figure 1, the complex park instance is not highly constrained, i.e. machines are not highly utilized, and the problem is therefore simpler than the instances with only 2 machines per face. This can also be seen in the aggregated results, where randomized restarts for this class of instances does not bring significant improvements.

Finally, we are only able to solve the smallest instances, *5F1C1M* and *5F1C2M*, to optimality. Figure 2 shows how the objective function evolves with solution time for larger problem instances, and it motivates the timeouts chosen. No significant improvements were seen when extending the timeout further (up to an hour).

5 Concluding Remarks

In this paper, we presented, to the best of the authors' knowledge, the first CP-based model for underground mine scheduling with promising preliminary results. The model resembles a flow-shop problem with the addition of periodic unavailabilities, after-lags, and a mix of interruptible and non-interruptible activities. Ongoing and future work includes: the integration of travel times for the machine park, replanning based on previous solutions to minimize schedule disruptions, personnel assignment and rostering. We would also like to explore techniques such as Large Neighborhood Search, and possibly decomposition approaches where the machine allocation problem and the scheduling are solved in a Logical Benders decomposition framework.

Bibliography

[1] Mincom. *Annual Study: Mining Executive Insights 2011*. Denver, CO, 2011.

[2] Marco Schulze, Julia Rieck, Cinna Seifi, and Jürgen Zimmermann. Machine scheduling in underground mining: an application in the potash industry. *OR Spectrum*, 38(2):365–403, 2016.

[3] Zhen Song, Håkan Schunnesson, Mikael Rinne, and John Sturgul. Intelligent scheduling for underground mobile mining equipment. *PloS one*, 10(6):e0131003, 2015.

[4] M Nehring, E Topal, and P Knights. Dynamic short term production scheduling and machine allocation in underground mining using mathematical programming. *Mining Technology*, 119(4):212–220, 2010.

[5] M Nehring, Erkan Topal, and J Little. A new mathematical programming model for production schedule optimization in underground mining operations. *Journal of the Southern African Institute of Mining and Metallurgy*, 110(8):437–446, 2010.

[6] Michelle Blom, Adrian R Pearce, and Peter J Stuckey. Short-term scheduling of an open-pit mine with multiple objectives. *Engineering Optimization*, 49(5):777–795, 2017.

[7] Masoumeh Mansouri. *A Constraint-Based Approach for Hybrid Reasoning in Robotics*. PhD thesis, Örebro university, 2016.

[8] Paul Saayman, IK Craig, and FR Camisani-Calzolari. Optimization of an autonomous vehicle dispatch system in an underground mine. *Journal of the Southern African Institute of Mining and Metallurgy*, 106(2):77, 2006.

[9] Mathieu Beaulieu and Michel Gamache. An enumeration algorithm for solving the fleet management problem in underground mines. *Computers & operations research*, 33(6):1606–1624, 2006.

[10] Michel Gamache, Renaud Grimard, and Paul Cohen. A shortest-path algorithm for solving the fleet management problem in underground mines. *European journal of operational research*, 166(2):497–506, 2005.

[11] Michael Pinedo. *Scheduling*. Springer, 2015.

[12] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2012.

[13] Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS conference*, pages 555–560, 2008.

[14] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 228–243, 2012.