Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Optimization Strategies for Resource-Constrained Project Scheduling Problems in Underground Mining

Alessandro Hill
Department of Industrial and Manufacturing Engineering, California Polytechnic State University, USA, ahill29@calpoly.edu, ime.calpoly.edu/faculty/ahill29

Andrea J. Brickey
Mining Engineering and Management, South Dakota School of Mines and Technology, USA, andrea.brickey@sdsmt.edu, https://www.sdsmt.edu/Directories/Personnel/Profile/Brickey,-Andrea/

Italo Cipriano
Alicanto Labs, Universidad Adolfo Ibáñez, Chile, italo.cipriano.j@uai.cl

Marcos Goycoolea
School of Business, Universidad Adolfo Ibáñez, Chile, marcos.goycoolea@uai.cl, http://mgoycool.uai.cl/

Alexandra Newman
Department of Mechanical Engineering, Colorado School of Mines, USA, newman@mines.edu, https://people.mines.edu/anewman/

Effective computational methods are important for practitioners and researchers working in strategic underground mine planning. We consider a class of problems that can be modeled as a resource-constrained project scheduling problem with optional activities; the objective maximizes net present value. We provide a computational review of math programming and constraint programming techniques for this problem, describe and implement novel problem-size reductions, and introduce an aggregated linear program that guides a list scheduling algorithm running over unaggregated instances. Practical, large-scale planning problems cannot be processed using standard optimization approaches. However, our strategies allow us to solve them to within about 5% of optimality in several hours, even for the most difficult instances.

Key words: Underground mine planning, resource-constrained project scheduling, constraint programming, mathematical programming
History:

1. Introduction and Literature Review

The resource-constrained project scheduling problem (Rcpsp) (Johnson 1967, Talbot 1982, Patterson 1984, Kolisch et al. 1995) seeks to optimally schedule jobs (or activities) over time in such a way as to comply with precedence and resource-capacity constraints. Traditionally, these models have sought to minimize makespan, though the objective can be generalized. Each activity is associated with a set of predecessors and a duration. The start time of each activity must be no earlier than the end time of any of its predecessors. The duration of each activity is assumed to be constant. That is, it is independent of when the activity starts. Each time period during which an activity is being executed, it consumes a certain amount of one or more resources, which are limited and renewed each time period.

The Rcpsp is broadly used to model project scheduling problems. An important application of the Rcpsp arises in the context of underground mine planning, in which development and extraction over the life of the mining project must be scheduled. Underground mine planning problems differ from traditional Rcpsps in that it is not necessary to schedule all jobs, and the objective is to maximize discounted cash flows. Realistic problem instances in underground mine planning tend to have a job count ranging in the tens of thousands, making them significantly larger than those found in academic papers.

1.1. Related Work

Johnson (1967) first introduces the Rcpsp in the context of mining, and, more generally, Pritsker et al. (1969) do so in the context of project scheduling. It has received notable attention in the literature since the late 1990's, with applications in logistics, manufacturing, service operations, pharmaceuticals and transportation. An overview of existing models can be found in Hartmann and Briskorn (2022), and methodological advances are given in Artigues et al. (2008) and Schwindt and Zimmermann (2015). In the most common variant of the Rcpsp, henceforth Rcpsp-Ms, the objective is to minimize project makespan, i.e., the time required to perform all activities. In the Rcpsp-Dc, first proposed by Russell (1970), the objective is to maximize discounted cashflows. Both Rcpsp-Ms and Rcpsp-Dc assume that all activities must be completed.

Because the number of academic papers that study the Rcpsp is so vast, we discuss only a few representative articles that highlight the four main approaches applicable to Rcpsp-Dc.

- Branch and Bound: Reyck and Herroelen (1998) and Vanhoucke et al. (2001) use exact branch-and-bound algorithms based on solving, at each node of the tree, a relaxation obtained by omitting resource capacity constraints.
- Metaheuristics: Vanhoucke (2009), Vanhoucke (2010), and Leyman and Vanhoucke (2015) solve Rcpsp-Dc with scatter-search and genetic algorithms.
- Mathematical Programming: The most common integer programming formulation for Rcpsp is the so-called time-index formulation, originally proposed by Pritsker et al. (1969) for Rcpsp-Ms. This formulation easily generalizes, and, because it

defines a binary variable for each activity-time pair, solving even the LP relaxation can be difficult. Fisher (1973) embeds Lagrangian relaxation for Pritsker's formulation of Rcpsp-Ms in a branch-and-bound algorithm. Christofides et al. (1987) strengthen Pritsker's formulation for Rcpsp-Ms and propose a branch-and-bound algorithm based on Lagrangian relaxation and disjunctive-arcs. Möhring et al. (2003) use Lagrangian relaxation for Rcpsp-Ms, but instead of branching, employ a list-scheduling heuristic based on Alpha Points. Kimms (2001) combine Lagrangian relaxation and list-scheduling for Rcpsp-Dc. Gu et al. (2013) also solve instances of Rcpsp-Dc, combining Lagrangian relaxation, Alpha Points and constraint programming.

- Constraint Programming (CP) and SAT solvers: Laborie (2005) exploits minimal critical sets for Rcpsp-Ms in a constraint-programming approach. Schutt et al. (2009, 2013) improve this methodology by using a lazy clause generation CP approach. Berthold et al. (2010) combine integer-programming methods with CP. Schutt et al. (2012b) extend the lazy-clause generation scheme to Rcpsp-Dc, and present favorable results relative to those generated by contemporary metaheuristics. A simple implementation with SAT solvers can outperform traditional CP solvers for instances of Rcpsp-Ms (de Azevedo et al. 2021), though it is not clear if these results would extend to Rcpsp-Dc.

This literature concerning Rcpsp-Ms and Rcpsp-Dc primarily compares different algorithms on public, artificially generated, instances. The most common datasets used for Rcpsp-Ms are found in PSPlib (Kolisch and Sprecher 1997). A few papers consider data set RG300 (Debels and Vanhoucke 2007), comprised of larger problems. For Rcpsp-Dc, the most commonly used data sets are DC1 (Vanhoucke et al. 2001) and DC2 (Vanhoucke 2010), containing instances with 120 activities or fewer, with the exception of RG300, which extends the set to 300 activities and uses network topologies in the construction framework; this, and resource dependencies, make them notoriously difficult to solve (Vanhoucke et al. 2016). For example, at the time of this writing, many instances with just 90 activities in PSPlib cannot be solved at all. The results in Leyman and Vanhoucke (2015) are, to our knowledge, the best reported results to date for Rcpsp-Dc on benchmark problems, which are slightly better than those produced by the method of Gu et al. (2013). It should be noted, however, that the latter provides provable optimality gaps.

We focus on variants of the Rcpsp used for strategic mine planning, for which the interest is to compute long-term, e.g., multiple-decade, schedules to prescribe life-of-mine extraction of ore. To this end, the orebody of interest containing economic concentrations of, for instance, copper and/or gold, is subdivided into smaller, notional, three-dimensional rectangular units (constituting a *block model*), for which extraction times are determined. The goal in these problems is to maximize discounted cash flows, as in Rcpsp-Dc.

Open pit mine planning problems, henceforth Rcpsp-Op, possess activities that correspond to the extraction of blocks, and require a single time period to complete; there are precedence restrictions but no time lags between activities. This special case arises because blocks are usually homogeneous in their extraction requirements and falls outside of the scope of this paper. Rivera Letelier et al. (2020) present recent advances in the open-pit mine planning setting, and a corresponding literature review. Public benchmarking instances for Rcpsp-Op can be found in Espinoza et al. (2013).

Instances of underground mine planning, henceforth Rcpsp-Ug, differ from Rcpsp-Ms only in that there is not a strict requirement to extract all activities. In underground mining, activities are related to development, extraction (e.g., mining or hauling), and backfilling, which is used to create stability after an area has been mined. The sequencing of these activities, in addition to their size and duration, is based on geologic characteristics, desired production rates, safety regulations, and economic factors. In practice, instances tend to be very large, possessing thousands to tens of thousands of activities. To date, few papers study Rcpsp-Ug. Gu et al. (2012) combine Lagrangian relaxation with a list-scheduling algorithm to solve real-world instances of Rcpsp-Ug ranging in size from $1,400$ to $12,000$ jobs, and from $2,000$ to $8,000$ time periods. Solutions with gaps averaging 20% are reported, with many higher than 50%, and taking as much as 18 hours to solve. Brickey (2015) follows a similar approach, using the Bienstock-Zuckerberg algorithm (Bienstock and Zuckerberg 2010) to solve the LP relaxation of Rcpsp-Ug with ventilation constraints in the form of knapsacks, for problems with $24,000$ jobs and 730 time periods, resulting in gaps ranging from 14% to 18% after approximately 12 hours. O'Sullivan and Newman (2015) develop a heuristic for a problem with a complex set of precedence constraints, inter alia, while King et al. (2017b) reformulate and decompose the model to yield solutions to design a transition point between open pit and underground operations and determine a corresponding schedule. Muñoz et al. (2018) present a detailed adaptation of the Bienstock-Zuckerberg algorithm to underground planning, though only Rcpsp-Ms and Rcpsp-Op instances are tested. Brickey et al. (2020) describe a case study of a mining project in Nevada that encompasses 16,000 activities and 14,400 time periods; integrality gaps resulting from the use of the academic software Omp, whose algorithmic details are given in Chicoisne et al. (2012) and Muñoz et al. (2018), range from 1% to 5%, depending on the number of time periods considered. Ogunmodede et al. (2021) present a modification of this model with the inclusion of ventilation considerations and refrigeration decisions, i.e., when to activate refrigeration needed to create a hospitable working environment for miners, and the algorithm used by Brickey et al. (2020) is adapted to solve the modified problems. For surveys on mine planning, see Newman et al. (2010) and Chowdu et al. (2021).

1.2. Contribution and Paper Organization

We evaluate state-of-the-art optimization approaches to Rcpsp-Dc and Rcpsp-Ug as given in IBM (2018) (for constraint programming); Muñoz et al. (2018) and Brickey et al. (2020) (for math programming); and, described in §4 and §5 (for the improvements). These algorithms have been adapted for underground mine planning with real-world datasets, and we show how these approaches can be used to obtain practical schedules and tight bounds on the corresponding optimal objective function value. Our main contributions are: (i) we adapt a preprocessing scheme traditionally used in open-pit production scheduling to an underground mining setting in which we eliminate variables for the case in which not every activity must be completed during the scheduling horizon; (ii) similar to Newman and Kuchta (2007) but employed on a much larger scale for the specialized Rcpsp problem structure, we present two different time aggregation schemes that, when used in an intermediate step, allow us to solve the original (unaggregated) problem instances to near-optimality very quickly; (iii) we introduce extensions to traditional list-scheduling techniques, including LP-induced release dates and diversification, to obtain high-quality solutions; (iv) we use these techniques to solve large-scale, real-world underground mining production-scheduling instances with tightly coupled activity precedence in an operationally feasible amount of time; and, (v) we compare these techniques against constraint programming, which is often cited as a competitive alternative to mathematical programming for scheduling problems.

The remainder of this paper is structured as follows: Section 2 provides relevant definitions and notation. Section 3 presents the time-aggregated formulations. Section 4 discusses preprocessing strategies to reduce problem instance size. Section 5 explains the algorithms we use to solve the optimization problems. Section 6 demonstrates our strategies on a variety of industry datasets, and Section 7 concludes.

2. Basic Problem Formulation

Let $\mathcal{T}$ be a discrete time horizon, $\mathcal{T} = \{1, ..., T\}$. We are given a set of activities $\mathcal{J}$ and a set of resources $\mathcal{R}$ available in each time period, i.e., so-called "renewable resources" in the scheduling literature. Executing an activity $j \in \mathcal{J}$ is associated with a monetary value $p_j \in \mathbb{R}$, a duration $d_j \in \mathbb{Z}^+$, and a per-period resource utilization $u_{jr} \in \mathbb{R}^+ \; \forall \; r \in \mathcal{R}$. Each resource $r \in \mathcal{R}$ has a constant availability $U_r \in \mathbb{R}^+$ per time period. Furthermore, we are given an activity-on-node network represented by an acyclic digraph $\mathbf{H}$ with node set $\mathcal{J}$ corresponding to the set of activities, and arc set $\mathcal{A}$, representing generalized precedence relationships (see Figure 1), which require that if arc $(i, j) \in \mathcal{A}$, and if activity j is scheduled, then activity i must be scheduled as well. Each arc $(i, j) \in \mathcal{A}$ is associated with a lag $l_{ij} \in \mathbb{Z}_0^+$ which is used to model the amount of time that must lapse between the start of activity i and the start of activity j. In this way, if $l_{ij} = d_i$, then activity j can only start once activity i is finished. If $l_{ij} > d_i$, then a time must lapse between the completion of i and the start time of j. Finally, if $l_{ij} < d_i$, then both activities can run simultaneously. In general, if $l_{ij} < 0$, then activity i can even be allowed to start after j. This paper assumes that $l_{ij} \geq 0$.

We write $i \prec j$ if there exists a directed path from node i to node j in $\mathbf{H}$. Furthermore, if $i \prec j$, let $\hat{d}(i, j)$ represent the length of a longest path from i to j (as measured in cumulative lags). If $i \prec j$, we say that i is a *predecessor* of j, and j is a *successor* of i, respectively. A predecessor (successor) i is *direct* with respect to j if $(i, j) \in \mathcal{A}$ $((j, i) \in \mathcal{A})$.

A feasible schedule S for the considered RCPSP is a set of scheduled activities $\mathcal{J}^S \subseteq \mathcal{J}$ and activity start times $\underline{t}_j^S \in \mathcal{T} \; \forall j \in \mathcal{J}^S$ (or activity end times $\bar{t}_j^S \in \mathcal{T} \; \forall j \in \mathcal{J}^S$) such that:

(i) if activity j is scheduled, then each direct predecessor i is scheduled and starts at least l_{ij} time units before the start time of j, and

(ii) the consumption of each resource in each time period does not exceed its availability.

For a given discount factor $\delta \in (0, 1]$, the net present value of schedule S is defined as the sum of the discounted profits of the scheduled activities: $\sum_{j \in \mathcal{J}^S} p_j (1 + \delta)^{-\underline{t}_j^S}$. Our variant of the RCPSP accepts the empty schedule as feasible, and corresponds to $PS|prec|\sum C_j^F \beta^{C_j}$ according to the classification scheme in Brucker et al. (1999). This problem is known to be NP-hard (Blazewicz et al. 1983).

The following time-indexed formulation for the RCPSP, called a *by-formulation* or *step-formulation*, is known to yield the tightest linear programming-based

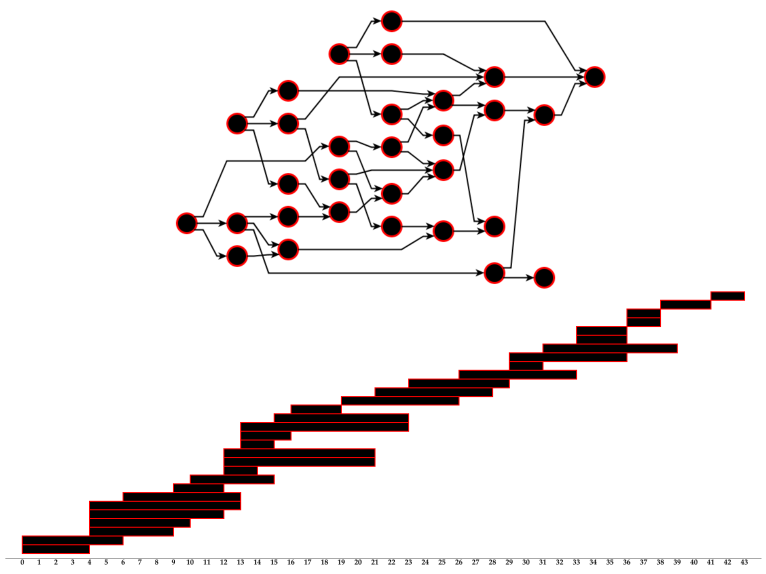

Figure 1 **The activity-on-node network for an RCPSP literature instance (j301-1 from Kolisch and Sprecher (1997)) with 30 activities (top) and an optimal schedule in Gantt chart representation (bottom).**

bounds (Artigues 2017). We define the following variables representing the time by which each activity starts, i.e., the activity starts in time t or before:

$$y_{jt} = \begin{cases} 1, & \text{if activity } j \text{ starts by time } t, \\ 0, & \text{otherwise.} \end{cases}$$

Then, the RCPSP can be written as follows:

$$(\mathcal{F}) \quad \max \quad \sum_{j \in \mathcal{J}} \sum_{t=1}^{T} \frac{p_j}{(1+\delta)^t} (y_{jt} - y_{j,t-1}) \tag{1a}$$

$$y_{jt} \le y_{j,t+1} \qquad j \in \mathcal{J}, 1 \le t \le T-1 \tag{1b}$$

$$\sum_{j \in \mathcal{J}: d_j \ge t} u_{jr} y_{jt} + \sum_{j \in \mathcal{J}: d_j < t} u_{jr} (y_{jt} - y_{j,t-d_j}) \le U_r \qquad r \in \mathcal{R}, 1 \le t \le T \tag{1c}$$

$$y_{j,t+l_{ij}} \le y_{it} \qquad (i,j) \in \mathcal{A}, 1 \le t \le T - l_{ij} \tag{1d}$$

$$y_{j0} = 0 \qquad j \in \mathcal{J} \tag{1e}$$

$$y_{jt} \in \{0,1\} \qquad j \in \mathcal{J}, 1 \le t \le T \tag{1f}$$

The objective given in (1a) maximizes the net present value over the scheduled jobs using discount rate δ. Variable linking inequalities (1b) ensure that if an activity starts by time t, it consequently starts by all the subsequent time periods. Resource-feasibility is enforced through knapsack inequalities (1c) in each period for each resource. Disaggregated *generalized precedence inequalities* (1d) ensure that the sequence and lag information given in $\mathcal{A}$ are appropriately considered; these collapse to precedence inequalities without lags if $l_{ij} = d_i$.

The bounds obtained by formulation $(\mathcal{F})$ can be replicated with a strengthened "at" formulation (Lambert et al. 2014), also called a *pulse-formulation* (Christofides et al. 1987, Espinoza et al. 2013), where the translation between the "by" variables and the "at" variables (which we denote x_{jt}) is given as follows: $x_{jt} = y_{jt} - y_{j,t-1}$. A tractability issue with formulation $(\mathcal{F})$ for underground mine planning arises owing to the nature of the application: a ten-year instance modeled at daily fidelity yields approximately 3,600 variables for each activity, where there can be several thousand activities. We therefore introduce two aggregate formulations that reduce problem size by considering a coarser time fidelity.

The Rcpsp can be specified for underground mine planning, as given by the mappings in Table 1; Terblanche (2017) presents background information, and we provide a more detailed description in §6.

3. Time-aggregated Formulations

We present both an approximate and an exact aggregation scheme; these are implemented by solving the corresponding integer program, whose resulting solutions help determine a near-optimal solution for the original model quickly relative to solution techniques available at the time of this writing.

3.1. The k-aggregated Model

Consider an instance of Rcpsp with T time periods. We describe how to define an instance with T/k, where k is an integer evenly dividing T. We begin with some definitions.

- Time periods: Define $\mathcal{T}^k = \{1, \ldots, T/k\}$.
- Durations: For each job $j \in \mathcal{J}$, let $d'_j = \lceil d_j/k \rceil$.
- Right-hand-side constants: For each $r \in \mathcal{R}$, define $U'_r = kU_r$.
- Resource consumption: For each $r \in \mathcal{R}$ and each $j \in \mathcal{J}$, let

$$u'_{jr} = \frac{u_{jr} d_j}{d'_j}.$$

- Discount rate: Define

$$\delta' = (1+\delta)^k - 1.$$

Note that this definition is such that:

Table 1 Model characteristics of the Rcpsp and corresponding examples for an underground mining application

Generic Rcpsp Characteristic	**UG Mine Planning Equivalent**
Activities: $j \in \mathcal{J}$	Development, extraction, transportation, backfilling
Durations: $d_j \ (j \in \mathcal{J})$	Hours to months required to execute each activity
Profits: $p_j \ (j \in \mathcal{J})$	Undiscounted value less cost associated with the completion of each activity
Resources: $r \in \mathcal{R}$	Finite resources consumed by activities, e.g., extraction capacity, volume of backfill paste
Capacities: $U_r \ (r \in \mathcal{R})$	Number of haulage tons or amount of paste consumable per time period
Resource utilization: u_{jr} $(j \in \mathcal{J}, r \in \mathcal{R})$	Number of extraction or mucking hours, or amount of paste required for a given backfill activity
Precedences: $(i,j) \in \mathcal{A}$	Extraction of a stope must precede backfill of said stope
Time lags: $l_{ij} \in \Re^+$	Duration of activities including any necessary lag between the completion of one activity and the start of another before extraction of a neighboring stope
Time horizon: $\mathcal{T}$	Number of hours to years considered in a single solve

$$\frac{1}{(1+\delta)^k} = \frac{1}{(1+\delta')}.$$

- Lags: For each $(i,j) \in \mathcal{A}$, let $l'_{ij} = \lfloor l_{ij}/k \rfloor$.

We can now formulate an aggregated version of problem $(\mathcal{F})$ using *by* variables w_{jt} as follows:

$$(\mathcal{F}^k) \quad \max \quad \sum_{j \in \mathcal{J}} \sum_{t=1}^{T/k} \frac{p_j}{(1+\delta')^t}(w_{jt} - w_{j,t-1}) \tag{2a}$$

$$w_{jt} \leq w_{j,t+1} \qquad j \in \mathcal{J}, 1 \leq t \leq T/k - 1 \tag{2b}$$

$$\sum_{j \in \mathcal{J}: d'_j \geq t} u'_{jr} w_{jt} + \sum_{j \in \mathcal{J}: d'_j < t} u'_{jr}(w_{jt} - w_{j,t-d'_j}) \leq U'_r \qquad r \in \mathcal{R}, 1 \leq t \leq T/k \tag{2c}$$

$$w_{j,t+l'_{ij}} \leq w_{it} \qquad (i,j) \in \mathcal{A}, 1 \leq t \leq T/k - l'_{ij} \tag{2d}$$

$$w_{j0} = 0 \qquad j \in \mathcal{J} \qquad (2e)$$

$$w_{jt} \in \{0,1\} \qquad j \in \mathcal{J}, 1 \le t \le T/k \qquad (2f)$$

Formulation ($\mathcal{F}^k$) corresponds to an instance of RCPSP; for each $j \in \mathcal{J}$ and $t \in \mathcal{T}^k$, w_{jt} represents starting job j no later than in aggregated time period t, or, with respect to the original unaggregated time periods, no later than in time period kt.

Note that, to obtain the aggregated data, we round the durations up and the lags down. In the former case, it would be highly unrepresentative to have an activity with a duration of zero, no matter how small the duration might be relative to the length of the aggregated time period. In the latter case, this direction of rounding offsets the rounded-down durations. To mitigate the number of activities that may be feasibly executed in a particular time period given the shortened values for the lags, thereby improving the quality of the approximation, we add arcs to the precedence graph associated with formulation ($\mathcal{F}^k$). We call these arcs strengthened and define them as follows. For each pair of activities $i, j \in \mathcal{J}$ such that $i \prec j$, define a precedence (i,j) in ($\mathcal{F}^k$), with lag:

$$l'_{ij} = \left\lfloor \frac{\hat{d}(i,j)}{k} \right\rfloor$$

where $\hat{d}(i,j)$ is the weight of the longest directed path from i to j in the precedence graph, and i is a predecessor of j. We define $\hat{\mathcal{A}}$ as the set of all strengthened arcs, and add the following precedence constraints to ($\mathcal{F}^k$):

$$w_{j,t+l'_{ij}} \le w_{it} \quad (i,j) \in \hat{\mathcal{A}},\ 1 \le t \le T/k - l'_{ij}.$$

In practice, this may induce a lot of arcs, but most of these include redundancies that can be eliminated with a weighted transitive closure, described in Section 4. To motivate the strengthening associated with arc addition, consider a simple example in which a set of activities are contained in a path of the precedence graph, and in which the lag of every associated arc is 2. Assume, for simplicity, that these activities do not consume any resources. Because of the lags, the i-th activity in the path can be scheduled, at the earliest, in time period $2i+1$. So, if $i = 7$, the earliest the activity would be scheduled would be in $t = 15$.

Suppose that we approximate this instance ($\mathcal{F}$) with ($\mathcal{F}^4$). The lag of every arc in this aggregated problem is $l'_{ij} = \lfloor l_{ij}/k \rfloor = \lfloor \frac{2}{4} \rfloor = 0$, signifying that, in ($\mathcal{F}^4$), all activities could be scheduled in the first time period, clearly not the case in ($\mathcal{F}$) and therefore a gross relaxation. By contrast, in the strengthened version of $(\mathcal{F}^4)$, the i-th activity could be scheduled, at the earliest, in the aggregated time period $t' = \lfloor \frac{i}{2} \rfloor$. Re-examining the particular case of $i = 7$, the earliest time period in which the strengthened version of $(\mathcal{F}^4)$ would schedule this activity would be in aggregated time period $t' = 3$. This would correspond to the interval $9,\ldots,12$ in $(\mathcal{F})$'s horizon, which is the interval prior to the one containing 15, a much better approximation of the true early start for $i = 7$.

3.2. The Safe k-aggregated Model

Formulation $(\mathcal{F}^k)$ is not a relaxation of $(\mathcal{F})$; as a consequence, for $k > 1$, the optimal objective function value of $(\mathcal{F}^k)$ does not necessarily provide an upper bound on the optimal value of $(\mathcal{F})$. We can modify $(\mathcal{F}^k)$, see, e.g., Carrasco et al. (2013), Carrasco et al. (2018), and Fuentes et al. (2021), to obtain a relaxation as follows. Define $(\mathcal{F}^{sk})$ by substituting constraints (2c) with:

$$\sum_{j \in \mathcal{J}: d'_j < t} d_j u_{jr} w_{j,t-d'_j} \leq tkU_r = tU'_r \qquad r \in \mathcal{R}, t \in \mathcal{T}^k, \tag{3}$$

and, by substituting, for each $j \in \mathcal{J}$, the objective function coefficient p_j by:

$$p'_j = \begin{cases} p_j \frac{(1+\delta')}{(1+\delta)} & \text{if } p_j > 0, \\ p_j & \text{otherwise.} \end{cases}$$

This definition can be interpreted as positive-valued activities are scheduled at the beginning of each aggregated time period, and negative-valued activities at the end.

Proposition 1. Let y be a feasible solution of $(\mathcal{F})$. Let $\mathcal{T}^k = \{1,\ldots,T/k\}$. For each $j \in \mathcal{J}$ and $t' \in \mathcal{T}^k$, let $w_{jt'} = y_{j,t'k}$. Solution w is feasible for $(\mathcal{F}^{sk})$. Moreover, the objective function value provided by w evaluated in $(\mathcal{F}^{sk})$ is higher than that provided by y evaluated in $(\mathcal{F})$.

Proof. By definition, we have $y_{i0} = 0$ for $i \in \mathcal{J}$; hence, $w_{i0} = 0$ for $i \in \mathcal{J}$.

- To determine feasibility, we evaluate the solution w in each of the constraints of $(\mathcal{F}^{sk})$:

  Consistency constraints

  $$w_{jt'} - w_{j,t'+1} = y_{j,t'k} - y_{j,(t'+1)k} \leq 0 \quad \forall j \in \mathcal{J}, t' \in \{1,...,T/k-1\}$$

  Precedence constraints

  Given $(i,j) \in \mathcal{A} \cup \hat{\mathcal{A}}$ it is necessary to prove:

  $$w_{jt'} - w_{i,t'-l'_{ij}} \leq 0 \quad \forall t' \text{ such that } l'_{ij} + 1 \leq t' \leq T/k.$$

— Case 1: $(i,j) \in \hat{\mathcal{A}}$. That is, $l'_{ij} = \left\lfloor \frac{\hat{d}(i,j)}{k} \right\rfloor$ and $l'_{ij} + 1 \leq t' \leq T/k$.

In this case:

$$kl'_{ij} \leq \hat{d}(i,j) \Rightarrow t'k - kl'_{ij} \geq t'k - \hat{d}(i,j) \Rightarrow y_{i,t'k-l'_{ij}k} \geq y_{i,t'k-\hat{d}(i,j)}$$

where $y_{i,t'k-\hat{d}(i,j)}$ is well defined because

$$t'k - \hat{d}(i,j) \geq \left\lfloor \frac{\hat{d}(i,j)}{k} \right\rfloor k - \hat{d}(i,j) + k \geq 0.$$

— Case 2: $(i,j) \in \mathcal{A}$. That is, $l'_{ij} = \left\lfloor \frac{l_{ij}}{k} \right\rfloor$ and $l'_{ij} + 1 \leq t' \leq T/k,$.

In this case:

$$w_{jt'} - w_{i,t'-l'_{ij}} = y_{j,t'k} - y_{i,k(t'-l'_{ij})} \leq y_{j,t'k} - y_{i,kt'-l_{ij}} \leq 0$$

where $y_{i,kt'-l_{ij}}$ is well defined because

$$kt' - l_{ij} \geq k(l'_{ij}+1) - l_{ij} = k \left\lfloor \frac{l_{ij}}{k} \right\rfloor - l_{ij} + k \geq 0.$$

<u>Capacity constraints</u>

First, observe that given $r \in \mathcal{R}$ and $1 < t \leq T/k$,

$$\sum_{j \in \mathcal{J}: d'_j < t} d_j u_{jr} w_{j,t-d'_j} = \sum_{j \in \mathcal{J}: d'_j < t} d_j u_{jr} y_{j,(t-d'_j)k} \leq \sum_{j \in \mathcal{J}: d_j < tk} d_j u_{jr} y_{j,tk-d_j}$$

The last inequality follows from the fact that for $j \in \mathcal{J}, d'_j < t \leq T/k$

$$d'_j = \left\lceil \frac{d_j}{k} \right\rceil \Rightarrow kd'_j \geq d_j \Rightarrow tk - kd'_j \leq tk - d_j \Rightarrow y_{j,tk-d'_jk} \leq y_{j,tk-d_j}$$

From the definition of y, for $r \in \mathcal{R}$ and $1 \leq t' \leq T$,

$$\sum_{j \in \mathcal{J}: d_j \geq t'} u_{jr} y_{jt'} + \sum_{j \in \mathcal{J}: d_j < t'} u_{jr}(y_{jt'} - y_{j,t'-d_j}) \leq U_r.$$

Then, assuming $tk - d_j > 0$ and summing on $t' \in \{1, \ldots, tk\}$

$$\sum_{t'=1}^{tk} \left(\sum_{j \in \mathcal{J}: d_j \geq t'} u_{jr} y_{jt'} + \sum_{j \in \mathcal{J}: d_j < t'} u_{jr}(y_{jt'} - y_{j,t'-d_j}) \right) \leq tkU_r,$$

where the left-hand side of the previous inequality can be written as

$$\sum_{j \in \mathcal{J}} u_{jr} \left(\sum_{t'=d_j+1}^{tk} (y_{jt'} - y_{j,t'-d_j}) + \sum_{t'=1}^{d_j} y_{jt'} \right) = \sum_{j \in \mathcal{J}} u_{jr} \sum_{t'=tk-d_j+1}^{tk} y_{jt'}.$$

By definition of y and the assumption $tk - d_j > 0$ we obtain that

$$\sum_{t'=tk-d_j+1}^{tk} y_{jt'} \geq d_j y_{j,tk-d_j},$$

and therefore

$$\sum_{j\in\mathcal{J}} u_{jr} \sum_{t'=tk-d_j+1}^{tk} y_{jt'} \geq \sum_{j\in\mathcal{J}:tk-d_j>0} u_{jr} d_j y_{j,tk-d_j},$$

and by the first observation, we finally conclude that:

$$\sum_{j\in\mathcal{J}:d'_j<t} d_j u_{jr} w_{j,t-d'_j} \leq tkU_r \quad \forall r \in \mathcal{R}, t \in \mathcal{T}^k$$

- We now prove the statement regarding the relative objective function values. Let j be arbitrary and fixed, and assume that for variable $y_{j\tau}$ the transition from 0 to 1 occurs at some period $\tau \in \{k(t-1)+1,\dots,kt\}$, where $t \in \{1,\dots,T/k\}$. The contribution of y_{jt} to the objective function of $(\mathcal{F})$ is a value in $\left\{\frac{p_j}{(1+\delta)^{k(t-1)+1}},\dots,\frac{p_j}{(1+\delta)^{kt}}\right\}$. On the other hand, the contribution of w_{jt} to the objective function of $(\mathcal{F}')$ equals $\frac{p'_j}{(1+\delta')^t} = \frac{p'_j}{(1+\delta)^{kt}}$. Therefore, it suffices to prove that: $\frac{p'_j}{(1+\delta)^{kt}} \geq \max\left\{\frac{p_j}{(1+\delta)^{t'}} : t' \in \{k(t-1)+1,\dots,kt\}\right\}$. If $p_j \leq 0$, then $\max\left\{\frac{p_j}{(1+\delta)^{t'}} : t' \in \{k(t-1)+1,\dots,kt\}\right\} = \frac{p_j}{(1+\delta)^{kt}}$, and, in this case, it suffices to consider $p'_j = p_j$. If $p_j > 0$, then $\max\left\{\frac{p_j}{(1+\delta)^{t'}} : t' \in \{k(t-1)+1,\dots,kt\}\right\} = \frac{p_j}{(1+\delta)^{k(t-1)+1}}$, and, in this case, we must consider p'_j such that $\frac{p'_j}{(1+\delta)^{kt}} \geq \frac{p_j}{(1+\delta)^{k(t-1)+1}}$. We conclude that $p'_j \geq \frac{p_j}{(1+\delta)^{1-k}} = p_j\frac{(1+\delta')}{(1+\delta)}$. The definition of p'_j satisfies this inequality, which concludes the proof. ■

4. Preprocessing Techniques

Model instance size significantly affects solvability because it is inextricably intertwined with the number of variables and constraints in formulation $(\mathcal{F})$ or any of its variants. To this end, we introduce a variety of techniques to reduce the number of activities $|\mathcal{J}|$ and arcs $|\mathcal{A}|$. Some of these techniques are simple, but they all significantly reduce problem size and expedite solutions in practice.

4.1. Elimination of Trivial Activities

We can eliminate each zero-profit activity $j \in \mathcal{J}$ (i.e., $p_j = 0$) if $u_{jr} = 0$ for all $r \in \mathcal{R}$. These types of activities are commonly added by mine planning schedulers as markers during the design phase. When eliminating an activity $j \in \mathcal{J}$, it is important to add an arc connecting each direct predecessor of j to each direct successor of j with the corresponding lags. That is, for every pair of arcs (i, j) and (j, ℓ) in $\mathcal{A}$, add an arc (i, ℓ)

with lag $l_{ij} + l_{j\ell}$. Then, after obtaining a solution without these activities, we post-process them in such that precedence is satisfied. (Clearly, the resource constraints are satisfied by default after these activities' re-introduction.)

4.2. Earliest Activity Start Times, and Unreachable Arcs and Nodes

For each activity $j \in \mathcal{J}$, we can compute a lower bound $\texttt{es}(j)$ on the earliest time it would be possible to start its execution, as follows:

$$\texttt{es}(j) = \max\{\hat{d}(i,j) : i \in \mathcal{J} : i \prec j\}.$$

We say that an activity j is *unreachable* if $\texttt{es}(j) > T$. Likewise, we say that a precedence arc is *unreachable* if any of its end-points is unreachable. Note that it is only necessary to define variables y_{jt} such that $t \geq \texttt{es}(j)$ and such that an activity is reachable. Because $\mathbf{H}$ is acyclic, $\texttt{es}(j)$ can be computed in linear time using a topological sorting of the nodes. If $j \in \mathcal{J}$ is such that $\texttt{es}(j) + d_j > T$, then activity j can be removed, because it is unreachable, i.e., impossible to complete it during the given time horizon.

4.3. Weighted Transitive Closure

Consider an arc $(i,j) \in \mathcal{A}$. If $\hat{d}(i,j) > l_{ij}$, then arc (i,j) can be removed from $\mathcal{A}$ without affecting the set of feasible solutions to the problem. We identify (and correspondingly remove) all such redundant arcs efficiently with complexity $\mathcal{O}(|\mathcal{J}||\mathcal{A}|)$, as follows:

1. Compute a topological ordering of the nodes $\mathcal{J}$, which can be accomplished in $\mathcal{O}(|\mathcal{J}| + |\mathcal{A}|)$, as first shown by Kahn (1962).
2. For each activity $i \in \mathcal{J}$, let $\mathcal{A}^+(i)$ be the set of all arcs with tail i. Test the redundancy of all arcs $(i,j) \in \mathcal{A}^+(i)$ as follows: By traversing the set of nodes in $\mathcal{J}$ in topological order, starting from i, compute the distance $\hat{d}(i,j)$ for all $j \succ i$. This can be done in $\mathcal{O}(|\mathcal{A}|)$. Then, compare each distance $\hat{d}(i,j)$ to l_{ij} to determine redundancy, which can be done in $\mathcal{O}(|\mathcal{J}||\mathcal{A}|)$.

We refer to the resulting precedence graph as the *weighted transitive closure* of the original precedence graph. The propagation of the (unweighted) transitive closure is known to be effective in CP solvers (Baptiste et al. 2012).

4.4. Final Contour

We adapt the notion of the *ultimate pit*, or the envelope of blocks which, in combination, are economical to remove in the context of open pit mine planning, to a more general setting in which the removal of blocks corresponds to the execution of jobs whose precedence is subject to lags. In this way, we eliminate variables associated with activities whose combined execution does not contribute positively to the objective function in any solution (Rivera Letelier et al. 2020). Note that this technique can be used only because jobs in RCPSP-UG are optional.

Consider the following relaxation of formulation $(\mathcal{F})$, which can be obtained by eliminating inequalities (1c):

$$
\begin{aligned}
(\underline{\mathcal{F}}) \quad \max \quad & \sum_{j\in\mathcal{J}}\sum_{t=1}^{T} \frac{p_j}{(1+\delta)^t}(y_{jt}-y_{j,t-1}) && && \text{(4a)}\\
y_{jt} \leq & \, y_{j,t+1} && j\in\mathcal{J}, 1\leq t\leq T-1 && \text{(4b)}\\
y_{j,t+l_{ij}} \leq & \, y_{it} && (i,j)\in\mathcal{A}, 1\leq t\leq T-l_{ij} && \text{(4c)}\\
y_{j0} = & \, 0 && j\in\mathcal{J} && \text{(4d)}\\
y_{jt} \in & \, \{0,1\} && j\in\mathcal{J}, 1\leq t\leq T && \text{(4e)}
\end{aligned}
$$

Observe that formulation $(\underline{\mathcal{F}})$ takes the form of a maximum closure problem (which has received much attention in the mine planning literature since the seminal paper of Lerchs and Grossmann (1965)):

$$\max\{cw : w_i \leq w_j \ \forall (i,j)\in\mathcal{A},\ w_i\in\{0,1\}\}. \tag{5}$$

This problem can be efficiently solved in polynomial time, and admits a unique, inclusion-wise minimal and optimal solution w^1, in the sense that all other optimal solutions w^* satisfying $w^1 \geq w^*$ are such that $w^1 = w^*$. For background on the maximum closure problem and a description of an algorithm that effectively solves it in practice, see Goldberg and Tarjan (1988), Cherkassky and Goldberg (1997), and Hochbaum (2008).

Proposition 2. Let y^1 be the optimal and inclusion-wise minimal solution of $(\mathcal{F})$, and y^2 any optimal solution of $(\underline{\mathcal{F}})$. For $m = 1, 2$, let $\mathcal{J}^m = \{j\in\mathcal{J} : y^m_{jT} = 1\}$. If T is sufficiently large, then $\mathcal{J}^1 \subseteq \mathcal{J}^2$.

Proof. Suppose, by contradiction, that the statement is false, and denote the non-empty set $\mathcal{J}^1 \setminus \mathcal{J}^2$ by $\mathcal{J}^\Delta$. Note that $\mathcal{J}^\Delta$ represents the set of activities scheduled in $\mathcal{J}^1$, but not in $\mathcal{J}^2$. We will show that the optimality of y^1 implies that the activities in $\mathcal{J}^\Delta$ must contribute positively to the objective function in problem $(\mathcal{F})$. This, however, leads to a contradiction, because if this were the case, then adding these activities to set $\mathcal{J}^2$ (possibly delaying their scheduled time) would have resulted in a strictly better solution than y^2 to $(\underline{\mathcal{F}})$.

To see that the activities in $\mathcal{J}^\Delta$ must contribute positively to the objective function in problem $(\mathcal{F})$, note first that if we restrict solution y_1 to only activities in $\mathcal{J}_1 \cap \mathcal{J}_2$, we obtain a feasible solution to $(\mathcal{F})$. In fact, by restricting y_1 to $\mathcal{J}_1 \cap \mathcal{J}_2$, the solution consumes no more of any resource types, and hence constraints (1c) must be satisfied. Moreover, since $\mathcal{J}_1 \cap \mathcal{J}_2$ is also a closure, precedences must be satisfied.

Because $\mathcal{J}^\Delta$ is non-empty, the optimality of y^1 implies that:

$$\sum_{j \in \mathcal{J}^\Delta} \sum_{t \geq 1} \frac{p_j}{(1+\delta)^t} (y^1_{jt} - y^1_{j,t-1}) \geq 0. \tag{6}$$

Otherwise, removing this part of the solution would yield a better one. Moreover, the inclusion-wise minimality of y^1 tells us that the inequality above is strict.

Now, for each $j \in \mathcal{J}$, $t \in \mathcal{T}$ such that $t \geq 1$, for a given k, define:

$$w^k_{j,t+k} = \begin{cases} y^1_{jt} & \text{if } j \in \mathcal{J}^\Delta \text{ and } t+k \in \mathcal{T} \\ 0 & \text{otherwise.} \end{cases}$$

Note that w^k is a partial solution that corresponds to scheduling activities in $\mathcal{J}^\Delta$ exactly k periods after they are scheduled in solution y^1.

From (6) we have, for all $k \geq 0$, that:

$$\sum_{j \in \mathcal{J}} \sum_{t \geq 1} \frac{p_j}{(1+\delta)^t} (w^k_{jt} - w^k_{j,t-1}) = \sum_{j \in \mathcal{J}^\Delta} \sum_{t \geq 1} \frac{p_j}{(1+\delta)^{t+k}} (y^1_{jt} - y^1_{j,t-1}) > 0. \tag{7}$$

Given that T is assumed to be large, there exists $k > 0$ such that $y^3 = y^2 + w^k$ is feasible for $(\underline{\mathcal{F}})$. In fact, it suffices to choose k so that all the predecessors of activities in $\mathcal{J}^\Delta$ have been completed in y^2. Note that in y^3: (i) all activities in $\mathcal{J}^1$ are scheduled exactly as in y^2; and (ii) all activities in $\mathcal{J}^\Delta$ are scheduled exactly as in y^1, but delayed by k periods. This, however, given (6), contradicts the optimality of y^2. Thus, we conclude that $\mathcal{J}^\Delta$ must be empty. ■

This proposition implies that we can compute y^2 and use it to eliminate each activity $j \in \mathcal{J} \setminus \mathcal{J}^2$ from the problem.

5. Optimization Algorithms

In this section, we describe various list scheduling algorithms that take as parameters two integers k_1 and k_2 that divide T, and proceed as follows: (i) execute the preprocessing steps described in Section 4 to reduce the size of $(\mathcal{F})$; (ii) solve the LP relaxations of

$(\mathcal{F}^k)$ and $(\mathcal{F}^{sk})$, generating a batch of feasible LP relaxation solutions to these problems; (iii) run a set of scheduling algorithms on the original, unaggregated problem $(\mathcal{F})$, using as input the LP relaxation solutions generated for problems $(\mathcal{F}^k)$ and $(\mathcal{F}^{sk})$. Finally, we describe the constraint programming procedure we use to contrast our implementations against a competitive methodology.

5.1. Solving the LP Relaxation

Solving the LP relaxation of models $(\mathcal{F})$, $(\mathcal{F}^k)$ and $(\mathcal{F}^{sk})$ can be difficult owing to the size of practical instances. In fact, time-indexed formulations of the Rcpsp grow linearly with the number of time periods, of which real-world problems tend to possess copious amounts, easily in the thousands. Decomposition algorithms address this difficulty; see, for example, Christofides et al. (1987), who use Lagrangian relaxation for Rcpsp-Ms. In order to solve the LP relaxation, we use the decomposition algorithm proposed by Muñoz et al. (2018), which is an extended version of the Bienstock-Zuckerberg (BZ) algorithm (Bienstock and Zuckerberg 2010) that incorporates activity durations, auxiliary variables and algorithmic speed-ups.

We run the Muñoz et al. (2018) implementation of the BZ algorithm twice: once to solve the LP relaxation of $(\mathcal{F}^k)$ and once to solve the LP relaxation of $(\mathcal{F}^{sk})$, and retain the best primal solution and best dual bound from each, respectively. In the case of $(\mathcal{F}^k)$, we additionally keep the primal solution found after every ten iterations. We use the best dual bound computed for $(\mathcal{F}^{sk})$ as a safe upper bound in order to compute an integrality gap, and hence, the quality of the solution generated. The primal solutions computed from $(\mathcal{F}^k)$ are used by the list scheduling algorithms, explained in the next section, in order to generate integer-feasible solutions.

5.2. Generation of an Integer-feasible Solution via List Scheduling

To obtain an integer-feasible solution to model $(\mathcal{F})$, we use a list-scheduling algorithm (Hall et al. 1997, Phillips et al. 1998), also employed in the context of underground mining (Gu et al. 2012, Brickey 2015, Muñoz et al. 2018). We propose a variant of this approach, in which we use a solution to the LP relaxation of aggregated problems $(\mathcal{F}^k)$ and $(\mathcal{F}^{sk})$ to obtain feasible solutions to the unaggregated problem $(\mathcal{F})$. This makes computing integer feasible solutions to $(\mathcal{F})$ significantly faster.

There are many different algorithmic variants of the list scheduling algorithm, the performance of which varies from problem to problem, and can depend on the LP relaxation solution used as input. We exploit these two types of performance variability as follows: (i) by using multiple LP solution iterates generated by the BZ algorithm which, by definition, are primal feasible – rather than just the final LP iterate; and, (ii) by running many variants of the list-scheduling algorithm on each of the aforementioned solutions. Exploiting performance variability has been successful in cutting plane generation, e.g., Fischetti et al. (2016), primal heuristics, e.g., Danna et al. (2005), and branching, e.g., Carvajal et al. (2014). See Lodi and Tramontani (2013) for a discussion. To our knowledge, this type of approach has not been applied to list scheduling. We next describe the mix of list scheduling variants that we use in our algorithm, first, by providing some definitions.

Definition 1. We define a *partial schedule* or *partial solution* of formulation $(\mathcal{F})$, as a pair $(\mathcal{S}, \mathbf{s})$, where $\mathcal{S} \subseteq \mathcal{J}$ indicates a set of selected jobs, and where s_j for each $j \in \mathcal{S}$ is an element of the vector $\mathbf{s}$ denoting the start time of job j. We say that a partial schedule $(\mathcal{S}, \mathbf{s})$ is *feasible* with respect to $(\mathcal{F})$ if the corresponding schedule satisfies precedence and resource consumption constraints.

Definition 2. Consider an instance $(\mathcal{F})$ of Rcpsp, a partial schedule $(\mathcal{S}, \mathbf{s})$, and a pair (t, j), with $t \in \mathcal{T}$, and $j \in \mathcal{J} \setminus \mathcal{S}$. We say that $(\bar{\mathcal{S}}, \bar{\mathbf{s}})$ *extends* $(\mathcal{S}, \mathbf{s})$ with (t, j) if $\bar{\mathcal{S}} = \mathcal{S} \cup \{j\}$, $\bar{s}_i = s_i$ for all $i \in \mathcal{S}$, and $\bar{s}_j = t$. We say that (t, j) *feasibly extends* $(\mathcal{S}, \mathbf{s})$ if $(\bar{\mathcal{S}}, \bar{\mathbf{s}})$ is a feasible partial schedule.

Definition 3. Given an instance $(\mathcal{F})$ of the Rcpsp, and a feasible partial solution $(\mathcal{S}, \mathbf{s})$ of $(\mathcal{F})$, let $\mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s})$ be the set of pairs $(t, j) \in \mathcal{T} \times \mathcal{J}$ that feasibly extends $(\mathcal{S}, \mathbf{s})$.

List scheduling algorithms start with an empty partial solution, and feasibly extend it, one iteration at a time. Algorithm 1 describes the general class of list scheduling algorithms that we consider.

Algorithm 1: List-Scheduling for Rcpsp with positive lags $(\mathcal{F})$

```
1: $\mathcal{S} \leftarrow \emptyset, \mathbf{s} \leftarrow \emptyset$
2: while $\mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s}) \neq \emptyset$ do
3:     Choose $(t, j) \in \mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s})$
4:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}, s_j \leftarrow t$
5: return $(\mathcal{S}, \mathbf{s})$
```

In executing Algorithm 1, it is critical to correctly choose, in each iteration, the pair (t, j) with which to extend the incumbent partial solution. In most list scheduling algorithms, this is controlled by two parameters:

- A priority rule vector π, of length $|\mathcal{J}|$. Given $i, j \in \mathcal{J}$, if $\pi_i > \pi_j$, then j is said to have higher priority than i.
- A sequencing rule: serial or parallel. When sequencing in parallel, only jobs that can be scheduled as early as possible are considered. When sequencing in serial, only jobs with the highest priority are considered.

A recent paper by Rivera Letelier et al. (2020) also proposes using a release date criterion in the context of Rcpsp-Op. That is, a vector ρ, of length $|\mathcal{J}|$, such that a pair $(t, j) \in \mathcal{E}((\mathcal{F}), \mathcal{S}, s)$ is selected only if $t \geq \rho_j$. To our knowledge, this has not been attempted for other classes of Rcpsp problems.

Algorithm 2 and Algorithm 3 formally describe the serial and parallel variants of Algorithm 1, respectively, in such a way as to incorporate the criteria established above.

Algorithm 2: Serial List Scheduling $((\mathcal{F}), \pi, \rho)$

```
1: $\mathcal{S} \leftarrow \emptyset, \mathbf{s} \leftarrow \emptyset$
2: while $\mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s}) \neq \emptyset$ do
3:     $j \leftarrow \text{argmin}\{\pi_i : (t', i) \in \mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s}), t' \in \mathcal{T}, i \in \mathcal{J}\}$
4:     $t \leftarrow \min\{t' : (t', j) \in \mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s})$ and $t' \geq \rho_j\}$
5:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}, s_j \leftarrow t$
6: return $(\mathcal{S}, s)$
```

Algorithm 3: Parallel List Scheduling $((\mathcal{F}), \pi, \rho)$

```
1: $\mathcal{S} \leftarrow \emptyset, \mathbf{s} \leftarrow \emptyset$
2: while $\mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s}) \neq \emptyset$ do
3:     $t \leftarrow \min\{t' : (t', i) \in \mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s})$ and $t' \geq \rho_i\}$
4:     $j \leftarrow \text{argmin}\{\pi_i : (t, i) \in \mathcal{E}((\mathcal{F}), \mathcal{S}, \mathbf{s})$ and $i \in \mathcal{J}\}$
5:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}, s_j \leftarrow t$
6: return $(\mathcal{S}, \mathbf{s})$
```

Given a solution $y^* \in [0, 1]^{\mathcal{J} \times \mathcal{T}^k}$ from the LP relaxation of $(\mathcal{F}^k)$, we generate priority rules π and release dates ρ by first computing the following values for each job.

- Expected Time. For each $j \in \mathcal{J}$, let:

$$\mathrm{XP}_j = \sum_{t=1}^{T/k} t y^*_{jt} + \frac{T}{k}\left(1 - \sum_{t=1}^{T/k} y^*_{jt}\right).$$

- Alpha Points. Given a fixed value $\alpha \in (0,1)$, for each $j \in \mathcal{J}$, let:

$$\mathrm{AP}_j(\alpha) = k \times \left(\min\left\{t : \sum_{t' \leq t} y^*_{jt'} \geq \alpha\right\} - 1\right).$$

Once computed, either XP or AP can be used as priority rules, for the latter of which different values of α result in different priority rules and release dates; the values of AP are elements of $\mathcal{T}$ and can be used directly as release dates.

Expected time was first used to define priority rules (π) by Chicoisne et al. (2012), and later by Rivera Letelier et al. (2020), in the context of open pit mine planning. It has also been used by Brickey (2015), King et al. (2017a,b), King and Newman (2018), Muñoz et al. (2018) and Brickey et al. (2020) in the context of underground mine planning. Alpha Points have been invoked in approximation algorithms for general classes of scheduling problems (Hall et al. 1997, Phillips et al. 1998). They have also been employed by Gu et al. (2012) in the context of underground mining. To our knowledge, the only appearance of release dates in list scheduling is by Rivera Letelier et al. (2020), who use alpha points for this purpose; all previous academic work on the RCPSP has used LP relaxation solutions of ($\mathcal{F}$) to compute alpha points and expected values, rather than LP relaxation solutions of ($\mathcal{F}^k$), or other aggregations, which are much faster to compute.

5.3. Running the List Scheduling Algorithm in Batches

Rather than run the list scheduling algorithm once, for a given input solution y^*, and a given priority rule and release date vector, we run the algorithm many times, using different input solutions, and different priority and release date vectors. The rules we use for a single y^* are as follows:

- We always use Alpha Points to define the release date of each activity. Specifically, we use a fixed alpha value of 0.01 and define:

$$\rho_j = \mathrm{AP}_j(0.01) \quad \forall j \in \mathcal{J}.$$

  This allows the postponement of activities in the final schedule that are also postponed in the LP solution.

- We consider two different strategies for defining priority rules:
  - Expected Strategy: We run the list scheduling algorithm using the expected times as a priority rule. That is: $\pi = XP$.
  - Alpha Strategy: We run the list scheduling algorithm using Alpha Points with fifty alpha values evenly spaced between zero and one. That is:

$$\pi_j^i = \mathrm{AP}_j(0.02i) \qquad \forall i \in 1, \ldots, 49, j \in \mathcal{J}$$

- For each priority rule used in both the Expected and Alpha Strategies, we run the list scheduling algorithm twice: once with parallel and once with serial sequencing, where we find empirically that there is not a dominating strategy.
- Upon completing each strategy, we run the post-optimization described in the previous section on the best solution found by the Parallel Algorithm 2 and on the best solution found by the Serial Algorithm 3.

Given a list of solutions $y^1, \ldots, y^m$ corresponding to different iterates of the BZ algorithm, we use the following rules on a single y^* as follows:

1. Final LP Iterate: We always run both the Alpha and Expected Strategies on the final LP iterate.
2. Best LP Iterate: For all iterates other than the final one, we first run the Expected Strategy and identify the iterate producing the solution with highest objective function value. The Alpha Strategy runs only on the iteration found by the Expected Strategy.

The reason for the execution of these rules in the manner prescribed by 1. and 2. is that we empirically find a strong correlation between the values resulting from the Alpha and Expected Strategies. Running the Alpha Strategy on all iterates produces only marginal improvements and is accompanied by significant run time.

5.4. Post Optimization

Consider an integer-feasible solution y of $(\mathcal{F})$. Let $\hat{\mathcal{J}} = \{j \in \mathcal{J} : y_{jT} = 1\}$. For each $j \in \hat{\mathcal{J}}$ let $t(j) = \min\{t : y_{jt} = 1\}$. Observe that $\hat{\mathcal{J}}$ corresponds to the set of jobs in $\mathcal{J}$ that are scheduled in solution y, and for each $j \in \hat{\mathcal{J}}$, $t(j)$ corresponds to the time in which j is scheduled.

It may be possible to improve solution y by simply removing some jobs from the schedule, and leaving all remaining jobs scheduled as they were. In order for such a new schedule to be feasible, it is necessary and sufficient that if a job is removed from the schedule, so must all of its successors be. The problem of deciding a best way to improve a solution in such a manner is simple to state, requiring a single, binary variable $w_{jt(j)}$ for each $j \in \hat{\mathcal{J}}$, indicating whether or not a job should remain as part of a solution. Both negative- and positive-valued jobs may be removed here if the overall contribution to the net present value is not worth their extraction. The integer program is run once, and can be stated as follows:

$$(\mathcal{PF}) \ \max \quad \sum_{j \in \hat{\mathcal{J}}} \frac{p_j}{(1+\delta)^t} w_{jt(j)} \tag{8a}$$

$$w_{jt(j)} \leq w_{it(j)} \quad \forall i,j \in \hat{\mathcal{J}}, (i,j) \in \mathcal{A} \tag{8b}$$

$$w_{jt(j)} \in \{0,1\} \quad \forall j \in \hat{\mathcal{J}} \tag{8c}$$

The objective function of this problem is the same as that of $(\mathcal{F})$. Constraints (8b) impose precedence, and constraints (8c) integrality. This is a maximum closure problem (Hochbaum 2008), and, as such, is easy to solve. Moreover, since the resulting solution satisfies all precedence constraints, and further, preserves feasibility with respect to the resource constraints, it is guaranteed to be feasible for $(\mathcal{F})$, and at least as good.

5.5. Constraint Programming

CP has been shown in a number of studies to be the most effective technique for solving small and difficult instances of the Rcpsp for both makespan minimization and net present value maximization; see, for example, Schutt et al. (2011, 2012a). Little has been published, however, on the effectiveness of CP for larger instances of the Rcpsp.

We adapt a CP formulation with makespan minimization (Laborie (2009), Laborie et al. (2018)) for our problem, $(\mathcal{F})$, that maximizes net present value. An interval variable y_j is used to represent each job $j \in \mathcal{J}$. Its start time, end time, and duration are accessed by $\texttt{start}(y_j)$, $\texttt{end}(y_j)$, and $\texttt{length}(y_j)$, respectively.

$$(\mathcal{F}_{CP}) \ \max \quad \underset{\mathrm{j}\in\mathcal{J}}{\texttt{sum}}\Big(e^{-\delta\cdot\texttt{end}(y_j)}p_j\Big) \tag{9}$$

$$\text{subject to} \quad \texttt{cumulative\_function}\big((y_1,u_{1r}),\ldots,(y_n,u_{nr})\big) \ \leq \ U_r \quad \forall r \in \mathcal{R}, \tag{10}$$

$$\texttt{end\_before\_start}(y_i, y_j, l_{ij}) \qquad \forall (i,j) \in \mathcal{A}, \tag{11}$$

$$y_j \ \texttt{interval variable of length} \ d_j \qquad \forall j \in \mathcal{J}. \qquad (12)$$

Formulation ($\mathcal{F}_{CP}$) maximizes the NPV of the schedule (9). Inequalities (10) ensure that the per-period capacity is not exceeded for any resource using a function which represents the cumulative resource usage by all jobs in each period. Both precedences and time lags are enforced by the generalized precedence constraints (11).

In the context of this study, we note that CP can be used both as an alternative means of generating an initial feasible solution to an instance of Rcpsp and as a means to improve a solution produced by some other method. We use the IBM ILOG CP Optimizer Version 12.8 (ILOG CP) on an appropriate CP model (Laborie et al. 2018) for both of these purposes, which iterates between using constraint programming and other techniques to compute a sequence of improving upper and lower bounds to the Rcpsp until optimality can be proven, or until a solution with a desired gap is found. Although the solver is effective at generating and improving the quality of feasible solutions, it is usually very slow in producing upper bounds that are stronger than those provided by the LP relaxation. Because ILOG CP does not allow the user to import externally computed upper bounds, we configure it to stop each time it generates a new and improving feasible solution. We then compare the quality of this solution against the upper bound provided by the LP solution. If the gap is below a pre-established value, we stop. Otherwise, ILOG CP continues attempting to improve the solution, or attains the desired gap with its own computed upper bounds. Laborie et al. (2013) show that providing initial solutions to a CP solver can be effective for smaller instances.

6. Computational Study

We conduct our computational experiments using Linux 2.6.32 as the operating system and an x86 64 architecture, with four eight-core Intel Xeon E5-2670 processors and 128 GB of RAM. All mining instances follow an underground method similar to the one displayed in Figure 2, in which ore is extracted one to three vertical levels at a time in an assigned direction, e.g., from top to bottom. Extraction of activities containing ore result in profit (as the recovered ore is processed and sold), while development and backfilling activities incur a cost but are necessary to access the ore. Equipment capacities, as well as labor availability, limit the mining rate and constitute the resource constraints. Our implementation of the BZ algorithm mimics the one presented in Muñoz et al. (2018), except that we update the version in the latter reference to IBM ILOG CPLEX, Version 12.8 (IBM 2018) for our master problems, and modify the code so as to save primal solutions every ten iterations. All list scheduling and arc aggregation algorithms are implemented and executed in Python 3.9 using the NumPy library; solution-time speed-ups could be realized by using an implementation in C or C++.

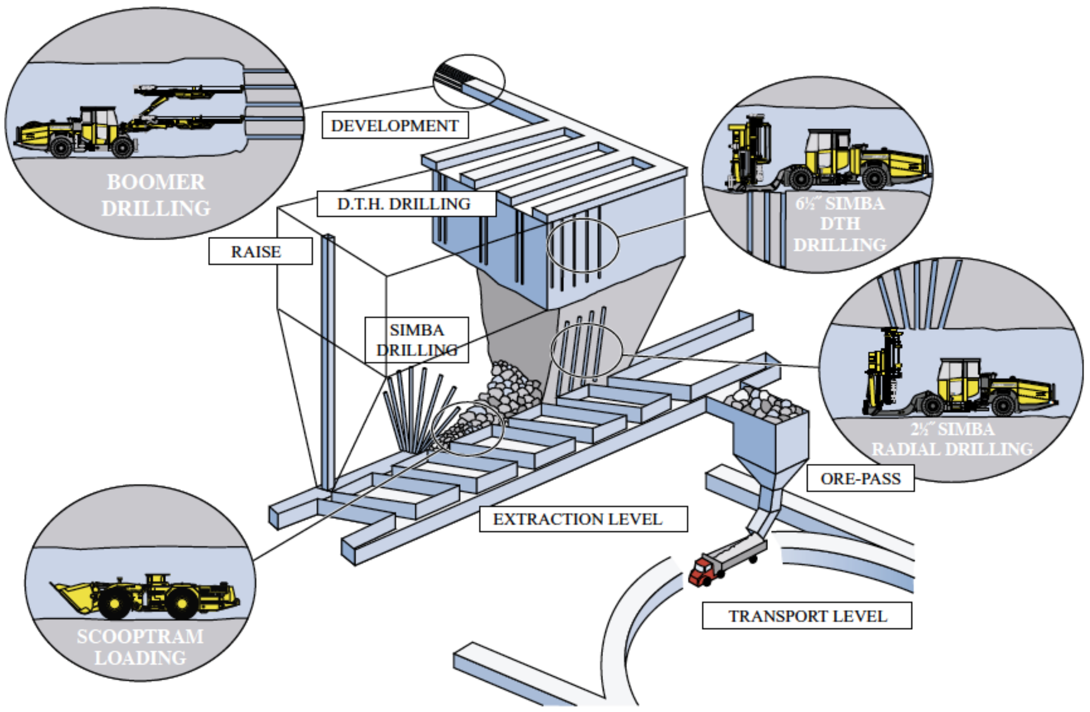

Figure 2 **A sublevel stoping mining operation, consisting of activities such as development and down-the-hole (DTH) drilling, in which ore is extracted from vertical levels (Hustrulid and Bullock 2001)**

We consider twelve strategic planning instances in our computational tests, nine of which (A1-A6 and B1-B3) correspond to instances with very few time periods and activities, henceforth referred to as *strategic-limited*. The final three data sets, (Catan, Dominion, and Agricola) referenced as *strategic-detailed*, correspond to instances with many time periods and activities. Strategic instances are used to evaluate the effect of different scenarios (e.g., cost structures, infrastructure, and major design decisions). Some underground mines are more uniform in layout, size, and production rates, thereby creating data sets with homogeneous activity durations that can be easily aggregated. Other underground mines have significant variations in these factors, leading to heterogeneous data with a wide range of fidelity. Because of this, many underground mining operations schedule on a daily basis, even for multi-year evaluations. Because a large number of scenarios must sometimes be evaluated, there is great value in producing schedules quickly.

For strategic instances in which the duration of an activity can range from a few hours to a few months because of the nature of the mine design, the number of activities

Table 2 Problem size, in terms of number of variables, activities and precedence, before and after preprocessing

	Number	Before All Preprocessing			After All Preprocessing		
Instance	of Periods	Variables	Activities	Precedences	Variables	Activities	Precedences
A1	60	87,377	1,598	38,016	71,192	1,296	4,295
A2	60	101,684	1,881	65,621	79,093	1,451	5,381
A3	60	104,580	1,944	67,898	80,003	1,473	5,575
A4	60	107,840	1,961	94,508	85,504	1,539	6,321
A5	60	116,302	2,140	71,583	89,541	1,625	7,450
A6	60	132,283	2,453	111,065	99,581	1,817	8,945
B1	30	24,479	1,157	104,754	24,287	1,157	3,629
B2	30	24,774	1,160	103,624	24,622	1,160	3,694
B3	30	25,339	1,176	110,801	25,293	1,176	3,834
Agricola	7,200	65,193,806	14,160	16,507	56,790,620	13,343	13,606
Catan	1,800	6,486,956	8,497	14,632	5,356,876	7,553	10,052
Dominion	3,600	65,405,074	28,883	49,313	55,382,828	28,393	29,788

can dramatically increase. All of the instances used in our study correspond to real planning problems with time fidelity of days. The names of the actual mines have been changed to protect confidentiality, and data has been scaled to prevent identification.

6.1. Preprocessing

Table 2 provides the sizes of our test instances, before and after preprocessing (see Section 4), as a function of the number of variables, the number of activities, and the number of precedences. The various preprocessing schemes have different effects on problem size reduction for the strategic-limited versus strategic-detailed instances. We observe that, on average, the number of variables in formulation ($\mathcal{F}$) can be reduced by 15.0%. The average reduction of problem activities and precedences is 13.0% and 77.1%, respectively. After preprocessing, the precedence graphs corresponding to the strategic-detailed instances are sparse, possessing fewer than two precedences per activity. By contrast, the strategic-limited instances contain four to five precedences.

6.2. Prior Methodologies

Table 3 provides results for the preprocessed instances with the best known solution methodologies in the academic literature prior to the publication of this paper. In order to compute solutions to the LP relaxation of ($\mathcal{F}$), we run the BZ algorithm (Muñoz et al. 2018) until the primal-dual gap falls below 10^{-4}, or stop after 15 days; the second column reports the corresponding amount of time. The best upper bound computed by the BZ algorithm after 24 hours of run time is reported in the third column. We report scaled objective function values by dividing the objective by the value of the best upper bound known for the corresponding problem, and multiplying by one hundred. The bounds of all problems, with the exception of Agricola, comes from solving the LP relaxation for up to 15 days (or until reaching the BZ stopping condition). The bound for Agricola comes from solving the safe LP bound to optimality (see Table 4). The fourth column reports the value of the solution obtained by using the TopoSort heuristic, as described in Brickey et al. (2020), taking as input the best primal solution found after 24 hours of running the BZ algorithm. This method corresponds to using a serial list scheduling algorithm, with expected times as a priority rule, and with no release dates. The penultimate column reports the value of the best primal feasible solution computed by the CP algorithm (IBM 2018) after 24 hours, starting from scratch. The final column provides the best upper bound computed by the CP algorithm at this same point in time.

The time taken to solve the LP relaxation of the strategic-limited instances is negligible, and the LP upper bounds appear to be tight; in any case, we are able to obtain a feasible solution using the TopoSort heuristic within about a few percentage points of optimality (the worst case being B2 at slightly more than 4%). Solutions of the linear programming relaxations for the strategic-detailed instances are more elusive in that two of the three instances require more than 15 days. Correspondingly, the integer-feasible solutions obtained via TopoSort using the best relaxed solutions after 24 hours demonstrate larger optimality gaps; it is unclear whether the bound is weaker or if the feasible solutions produced by the TopoSort algorithms are not as good. Overall, the mathematical programming results shown in this table are reasonable for the strategic-limited instances, where a solution within 5% of optimality can be obtained in seconds with the TopoSort heuristic. However, for the strategic-detailed instances, the best solutions require significant computational time to obtain, and the integer programming gaps are poorer (i.e., greater than 5%).

Attempting to ameliorate the poorer results for the strategic-detailed instances using constraint programming (CP) is not successful. The penultimate column of Table 3 shows that the quality of the primal solutions produced by the CP algorithm are erratic;

Table 3 **Time taken to solve the LP relaxation, and quality of the solutions obtained by using the TopoSort heuristic.** LP SOLVE TIME **indicates the time taken by the BZ algorithm to reach the termination conditions of a gap of** 10^{-4} **or a time limit of 15 days**[†]. LP24 UBOUND **indicates the best dual bound reported by the BZ algorithm after 24 hours of run time.** TOPO24 OBJVAL **indicates the objective function value of the TopoSort heuristic, when run on the best primal solution found by BZ after 24 hours. The final two columns,** CP24 OBJVAL **and** CP24 UBOUND, **provide the best objective function value and best upper bound computed by the CPLEX Constraint Programming solver in 24 hours. All objective function values are scaled dividing by the value of the best dual LP upper bound computed for the problem, and multiplying by one hundred.**

Instance	LP Solve Time[†]	LP24 Ubound	Topo24 Objval	CP24 Objval	CP24 Ubound
A1	7 s	100	98.7	99.2	121.1
A2	7 s	100	98.6	99.2	116.6
A3	8 s	100	97.8	99.1	133.7
A4	21 s	100	96.0	98.7	140.0
A5	23 s	100	97.2	98.0	163.4
A6	32 s	100	96.5	98.5	167.2
B1	2 s	100	96.6	86.4	304.6
B2	1 s	100	95.7	89.0	295.3
B3	2 s	100	96.6	92.7	284.4
Agricola	15 d[†]	107.3	94.5	91.0	186.1
Catan	5.1 d	100.0	92.6	93.4	121.6
Dominion	15 d[†]	102.2	92.7	71.9	281.8

while there is improvement for some of the already-well-performing strategic-limited instances (i.e., those in the A-SERIES), the strategic-detailed instances remain difficult. In fact, the list scheduling heuristic (based off the LP solution obtained within 24 hours of run time) produces significantly better results than CP (also run for 24 hours) for Agricola and Dominion. Moreover, the CP algorithm does not compute strong upper bounds; rather, the upper bounds afforded by the LP relaxation after the same amount of time are significantly better. We note that CP could potentially improve upon its current results using the TopoSort solution as input (Laborie et al. 2013). In fact, in revisiting the strategic-detailed instances, we can demonstrate significant improvement in tractability. In addition to pre-processing (where the size reductions are given in Table 2) and list scheduling which we implement in this section on the original instances, in the following section, we solve ($\mathcal{F}^k$) and ($\mathcal{F}^{sk}$) on the time-aggregated instances, which affords excellent performance when compared with that for the strategic-detailed instances in Table 3.

6.3. Improving the Solutions of the Strategic-detailed Problems

Table 4 reports, for each of the strategic-detailed instances, and for different values of k, the results obtained from using the methodology described in Section 5. Specifically, for each of the strategic-detailed instances, and for different values of k, we report: (i) the best LP upper bound computed by the BZ algorithm for $(\mathcal{F}^k)$ and $(\mathcal{F}^{sk})$ (third and fourth columns, respectively), (ii) the time required by the BZ algorithm, for $(\mathcal{F}^k)$ and $(\mathcal{F}^{sk})$, to reach the terminating condition (fifth and sixth columns, respectively), (iii) the value of the best feasible solution computed for $(\mathcal{F})$ by employing the methodologies described in Section 5.3, using as input the best primal solutions computed from $(\mathcal{F}^k)$ (seventh column), (iv) the total time taken to compute the list scheduling solutions (eighth column), and (v) the value of the best feasible solution computed by the CP algorithm after 24 hours, when providing as input the best solution computed by the list scheduling algorithm. The time reported in the penultimate column is the *serial* time; the actual time was much less because the multiple list scheduling runs can be completed independently in parallel on an eight-core machine, taking almost exactly one-eighth the time reported, which does not include $(\mathcal{F}^k)$'s LP relaxation solve. Thus, the total time required to compute a feasible solution is obtained by summing the values in the fifth and eighth columns. Finally, note that $(\mathcal{F}^1)$ is equivalent to $(\mathcal{F})$. In the case of Agricola, it can be seen that the LP upper bound computed from $(\mathcal{F}^{s1})$ is better than that of $(\mathcal{F}^1)$. This is likely because, after 15 days, $(\mathcal{F}^1)$ is still far from being solved by the BZ algorithm, and the best Lagrangian bound found with the stopping condition is still poor.

For all instances, solving the LP relaxation of $(\mathcal{F}^{sk})$ is significantly faster than solving that of $(\mathcal{F}^k)$, see Table 4. Furthermore, the amount of time required to solve either model decreases rapidly when increasing k. The LP upper bounds for $(\mathcal{F})$ are similar to those of $(\mathcal{F}^k)$, regardless of k. This suggests that the optimal LP relaxation value of $(\mathcal{F}^k)$ is a reasonable approximation to that of $(\mathcal{F})$. However, the safe LP bounds computed from $(\mathcal{F}^{sk})$ tend to be conservative. Finally, while the upper bounds provided by $(\mathcal{F}^{s1})$, in general, are good and valid, those provided by $(\mathcal{F}^{s60})$ are computed in negligible time and are only a few percentage points weaker.

In terms of the feasible solutions obtained for $(\mathcal{F})$ from the list-scheduling methodology, it can be seen that: (i) the quality of solutions is very good. In fact, for all but two instances (Agricola 30 and Agricola 60) the value is within 5%. Moreover, for Catan and Dominion, the value is within 4%. This is significantly better than the values seen in Table 3; (ii) the quality of the solutions obtained is very similar in all instances when k is less than or equal to 10. However, the time required to obtain these diminishes dramatically as k increases, because the time required to solve the LP relaxation of $(\mathcal{F}^k)$ seems to decrease exponentially with k, suggesting that $k = 10$ is a quick way to produce high-quality solutions; (iii) even when given the best list-scheduling solution as a warm start, the constraint programming algorithm yields little improvement. Nonetheless, in summary, the overall results are significantly better than those shown in Table 3, which were computed with the approach described in Brickey et al. (2020), both in terms of time and quality of solutions obtained.

Table 4 **Objective function value of approximate ($\mathcal{F}^k$) and safe ($\mathcal{F}^{sk}$) LP relaxations for different levels of time aggregation k, and a comparison with constraint programming, where we provide the value of the best feasible solution computed by the CP algorithm after 24 hours (+CP24), when providing as input the best solution computed by the list scheduling algorithm. All values are divided by the best known upper bound of the problem, and multiplied by one hundred. The List Scheduling time column shows the total time required to run all the list scheduling algorithms using different levels of aggregation for the strategic-detailed data sets. List Scheduling solve time does not include the time taken to solve the LP relaxation**

.

		LP Upper Bound		LP Solve Time [h]		List Scheduling		+CP24
Instance	k	Approximate	Safe	Approximate	Safe	Best val.	Time [h]	Best val.
Agricola	1	101.7	100.0	360.00	23.64	95.1	2.32	95.5
	2	99.6	101.9	153.14	6.75	95.0	6.65	95.2
	3	99.7	102.1	35.78	3.42	95.0	6.88	95.1
	5	99.7	102.4	8.32	1.28	95.2	4.12	95.5
	10	99.7	103.0	1.88	0.32	95.0	3.20	95.0
	30	99.9	103.6	0.31	0.06	94.5	2.74	94.6
	60	100.3	104.4	0.09	0.02	94.1	2.39	94.2
Catan	1	100.0	101.4	123.51	1.25	97.4	4.72	97.5
	2	100.0	101.4	4.75	0.18	97.5	3.11	97.6
	3	100.0	101.5	2.09	0.09	97.4	3.52	97.6
	5	100.0	101.6	0.70	0.04	97.3	3.21	97.4
	10	100.1	102.0	0.15	0.01	97.2	1.98	97.3
	30	100.3	102.5	0.01	0.00	96.6	1.28	96.7
	60	100.6	103.1	0.00	0.00	96.2	1.15	96.4
Dominion	1	100.0	100.1	360.00	61.67	98.0	26.85	98.0
	2	100.0	101.4	56.22	6.21	97.4	15.68	97.5
	3	100.0	101.5	23.16	2.95	97.6	14.54	97.7
	5	100.0	101.6	7.04	0.00	97.5	12.62	97.6
	10	100.1	101.8	1.93	0.20	97.6	13.15	97.6
	30	100.5	102.7	0.21	0.03	97.4	8.19	97.4
	60	101.0	103.9	0.00	0.01	96.8	7.25	97.0

7. Conclusion

Many strategic underground mine planning models possess a similar structure, that of a resource-constrained precedence problem, which we exploit in our mathematical programming approach. Constraint-programming techniques have also historically been used to find good, feasible solutions.

With preprocessing and new implementations of list-scheduling heuristics, we can significantly improve solution quality and reduce solution times for real-life mining instances when compared against both state-of-the-art mathematical programming and constraint programming techniques available at the time of this writing. Specifically, the instances we test can be solved to about 3% of optimality in a few minutes for the strategic-limited instances and to about 5% of optimality within several hours for the strategic-detailed instances. Conventional, untailored mathematical programming algorithms would require days, and even constraint programming solvers deliver solutions that may not be within 20% of optimality after the same amount of time.

Our results have implications for the ability of mine planners to examine many scenarios; future research would incorporate extended problem structures, e.g., those considering ventilation with the option of refrigeration, or minimizing makespan rather than maximizing net present value.

Acknowledgements

We acknowledge funding from Alford Mining Systems in partial support of this project. This work was also supported by Centro de Modelamiento Matemático (CMM), ACE210010 and FB21005, BASAL funds for center of excellence from ANID-Chile, and FONDEF Grant ID19—10164 by ANID-Chile. We thank Akshay Chowdu of the South Dakota School of Mining and Technology for his help in creating some of the instances and Levente Sipeki for performing preliminary computation. Powered@NLHPC: This research was partially supported by the supercomputing infrastructure of the NLHPC (ECM-02), in Santiago, Chile. Finally, the authors wish to thank two anonymous referees whose comments significantly improved the paper.

References

Artigues C (2017) On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters* 45(2):154–159.

Artigues C, Demassey S, Néron E (2008) *Resource-Constrained Project Scheduling. Models, Algorithms, Extensions and Applications.* Control Systems, Robotis and Manufacturing (John Wiley & Sons, Inc.).

Baptiste P, Le Pape C, Nuijten W (2012) *Constraint-based scheduling: Applying constraint programming to scheduling problems*, volume 39 (Springer Science & Business Media).

Berthold T, Heinz S, Lübbecke M, Möhring R, Schulz J (2010) A constraint integer programming approach for resource-constrained project scheduling. *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, 313–317 (Springer).

Bienstock D, Zuckerberg M (2010) Solving LP relaxations of large-scale precedence constrained problems. *International Conference on Integer Programming and Combinatorial Optimization*, 1–14 (Springer).

Blazewicz J, Lenstra JK, Kan AR (1983) Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics* 5(1):11–24.

Brickey A, Chowdu A, Newman A, Goycoolea M, Godard R (2020) Barrick's Turquoise Ridge gold mine optimizes underground production scheduling operations. Published online July 2020, *INFORMS Journal on Applied Analytics*, URL `http://dx.doi.org/https://doi.org/10.1287/inte.2020.1027`.

Brickey AJ (2015) *Underground production scheduling optimization with ventilation constraints*. Ph.D. thesis, Colorado School of Mines.

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1):3–41.

Carrasco RA, Iyengar G, Stein C (2013) Single machine scheduling with job-dependent convex cost and arbitrary precedence constraints. *Operations Research Letters* 41(5):436–441.

Carrasco RA, Iyengar G, Stein C (2018) Resource cost aware scheduling. *European Journal of Operational Research* 269(2):621–632.

Carvajal R, Ahmed S, Nemhauser G, Furman K, Goel V, Shao Y (2014) Using diversification, communication and parallelism to solve mixed-integer linear programs. *Operations Research Letters* 42(2):186–189.

Cherkassky B, Goldberg A (1997) On implementing the push—relabel method for the maximum flow problem. *Algorithmica* 19(4):390–410.

Chicoisne R, Espinoza D, Goycoolea M, Moreno E, Rubio E (2012) A new algorithm for the open-pit mine production scheduling problem. *Operations Research* 60(3):517–528.

Chowdu A, Nesbitt P, Brickey A, Newman A (2021) Operations research in underground mine planning: A review. Published online October 2021, *INFORMS Journal on Applied Analytics*, URL `http://dx.doi.org/https://doi.org/10.1287/inte.2021.1087`.

Christofides N, Alvarez-Valdés R, Tamarit J (1987) Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research* 29(3):262–273.

Danna E, Rothberg E, Le Pape C (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming* 102(1):71–90.

de Azevedo GHI, Pessoa AA, Subramanian A (2021) A satisfiability and workload-based exact method for the resource constrained project scheduling problem with generalized precedence constraints. *European Journal of Operational Research* 289(3):809–824.

Debels D, Vanhoucke M (2007) A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research* 55(3):457–469.

Espinoza D, Goycoolea M, Moreno E, Newman A (2013) Minelib: A library of open pit mining problems. *Annals of Operations Research* 206:93–114.

Fischetti M, Lodi A, Monaci M, Salvagnin D, Tramontani A (2016) Improving branch-and-cut performance by random sampling. *Mathematical Programming Computation* 8(1):113–132.

Fisher M (1973) Optimal solution of scheduling problems using Lagrange multipliers: Part I. *Operations Research* 21(5):1114–1127.

Fuentes D, Carrasco RA, Moreno E (2021) *Algorithms for resource-constrained project scheduling in underground mining operations*. Master's thesis, Industrial Engineering and Operations Research, Universidad Adolfo Ibáñez.

Goldberg A, Tarjan R (1988) A new approach to the maximum-flow problem. *Journal of the ACM* 35(4):921–940.

Gu H, Schutt A, Stuckey PJ (2013) A Lagrangian relaxation based forward-backward improvement heuristic for maximising the net present value of resource-constrained projects. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 340–346 (Springer).

Gu H, Stuckey PJ, Wallace MG (2012) Maximising the net present value of large resource-constrained projects. *International Conference on Principles and Practice of Constraint Programming*, 767–781 (Springer).

Hall L, Schulz A, Shmoys D, Wein J (1997) Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research* 22(3):513–544.

Hartmann S, Briskorn D (2022) An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 297(1):1–14.

Hochbaum DS (2008) The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research* 56(4):992–1009.

Hustrulid W, Bullock R (2001) *Underground mining methods: Engineering fundamentals and international case studies* (SME).

IBM (2018) IBM ILOG AMPL Version 12.8 User's Guide: Standard (Command-line) Version including CPLEX Directives.

Johnson TJR (1967) *An algorithm for the resource constrained project scheduling problem*. Ph.D. thesis, Massachusetts Institute of Technology.

Kahn AB (1962) Topological sorting of large networks. *Communications of the ACM* 5(11):558–562.

Kimms A (2001) Maximizing the net present value of a project under resource constraints using a Lagrangian relaxation based heuristic with tight upper bounds. *Annals of Operations Research* 102(1-4):221–236.

King B, Goycoolea M, Newman A (2017a) New integer programming models for tactical and strategic underground production scheduling. *Mining Engineering* 69(3).

King B, Goycoolea M, Newman A (2017b) Optimizing the open pit-to-underground mining transition. *European Journal of Operational Research* 257(1):297–309.

King B, Newman A (2018) Optimizing the cutoff grade for an operational underground mine. *Interfaces* 48(4):357–371.

Kolisch R, Sprecher A (1997) PSPLIB-a project scheduling problem library. *European Journal of Operational Research* 96(1):205–216.

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science* 41(10):1693–1703.

Laborie P (2005) Complete MCS-based search: Application to resource constrained project scheduling. *IJCAI*, 181–186.

Laborie P (2009) IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 148–162 (Springer).

Laborie P, Refalo P, Shaw P (2013) Model presolve, warmstart and conflict refining in CP optimizer. *CP Solvers: Modeling, Applications, Integration, and Standardization–CP2013 Workshop* (CP2013).

Laborie P, Rogerie J, Shaw P, Vilím P (2018) IBM ILOG CP Optimizer for scheduling. *Constraints* 23(2):210–250.

Lambert WB, Brickey A, Newman AM, Eurek K (2014) Open-pit block-sequencing formulations: a tutorial. *Interfaces* 44(2):127–142.

Lerchs H, Grossmann IF (1965) Optimum design of open-pit mines. *Transactions of the Canadian Institute of Mining* 68:17–24.

Leyman P, Vanhoucke M (2015) A new scheduling technique for the resource—constrained project scheduling problem with discounted cash flows. *International Journal of Production Research* 53(9):2771–2786.

Lodi A, Tramontani A (2013) Performance variability in mixed-integer programming. *Theory Driven by Influential Applications*, 1–12 (INFORMS).

Möhring R, Schulz A, Stork F, Uetz M (2003) Solving project scheduling problems by minimum cut computations. *Management Science* 49(3):330–350.

Muñoz G, Espinoza D, Goycoolea M, Moreno E, Queyranne M, Letelier OR (2018) A study of the Bienstock–Zuckerberg algorithm: Applications in mining and resource constrained project scheduling. *Computational Optimization and Applications* 69(2):501–534.

Newman AM, Kuchta M (2007) Using aggregation to optimize long-term production planning at an underground mine. *European Journal of Operational Research* 176(2):1205–1218.

Newman AM, Rubio E, Caro R, Weintraub A, Eurek K (2010) A review of operations research in mine planning. *Interfaces* 40(3):222–245.

Ogunmodede O, Lamas P, Brickey A, Bogin G, Newman A (2021) Underground production scheduling with ventilation and refrigeration considerations. Published online January 2022, *Optimization and Engineering*, URL `http://dx.doi.org/https://doi.org/10.1007/s11081-021-09682-4`.

O'Sullivan D, Newman A (2015) Optimization-based heuristics for underground mine scheduling. *European Journal of Operational Research* 241(1):248–259.

Patterson JH (1984) A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science* 30(7):854–867.

Phillips C, Stein C, Wein J (1998) Minimizing average completion time in the presence of release dates. *Mathematical Programming* 82(1-2):199–223.

Pritsker AAB, Waiters LJ, Wolfe PM (1969) Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science* 16(1):93–108.

Reyck BD, Herroelen W (1998) An optimal procedure for the resource-constrained project scheduling problem with discounted cash flows and generalized precedence relations. *Computers & OR* 25(1):1–17.

Rivera Letelier O, Espinoza D, Goycoolea M, Moreno E, Muñoz G (2020) Production scheduling for strategic open pit mine planning: A mixed-integer programming approach. *Operations Research* 68(5):1425–1444.

Russell A (1970) Cash flows in networks. *Management Science* 16(5):357–373.

Schutt A, Chu G, Stuckey P, Wallace M (2012a) Maximising the net present value for resource-constrained project scheduling. *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems* 362–378.

Schutt A, Chu G, Stuckey PJ, Wallace MG (2012b) Maximising the net present value for resource-constrained project scheduling. *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, 362–378 (Springer).

Schutt A, Feydy T, Stuckey PJ (2013) Explaining time-table-edge-finding propagation for the cumulative resource constraint. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 234–250 (Springer).

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2009) Why cumulative decomposition is not as bad as it sounds. *Principles and Practice of Constraint Programming-CP 2009*, 746–761 (Springer).

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2011) Explaining the cumulative propagator. *Constraints* 16(3):250–282.

Schwindt C, Zimmermann J (2015) *Handbook on Project Management and Scheduling.* International Handbooks on Information Systems (Springer International Publishing).

Talbot FB (1982) Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science* 28(10):1197–1210.

Terblanche S (2017) *Resource constrained project scheduling models and algorithms applied to underground mining.* Ph.D. thesis, Stellenbosch: Stellenbosch University.

Vanhoucke M (2009) A genetic algorithm for net present value maximization for resource constrained projects. Cotta C, Cowling P, eds., *Evolutionary Computation in Combinatorial Optimization*, 13–24 (Berlin, Heidelberg: Springer Berlin Heidelberg).

Vanhoucke M (2010) A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. *International Journal of Production Research* 48(7):1983–2001.

Vanhoucke M, Coelho J, Batselier J (2016) An overview of project data for integrated project management and control. *Journal of Modern Project Management* 3(3):6–21.

Vanhoucke M, Demeulemeester E, Herroelen W (2001) On maximizing the net present value of a project under renewable resource constraints. *Management Science* 47(8):1113–1121.