Interfaces
Vol. 44, No. 2, March–April 2014, pp. 127–142
ISSN 0092-2102 (print) | ISSN 1526-551X (online)

http://dx.doi.org/10.1287/inte.2013.0731
© 2014 INFORMS

Open-Pit Block-Sequencing Formulations: A Tutorial

W. Brian Lambert
Division of Economics and Business, Colorado School of Mines, Golden, Colorado 80401, wlambert@mines.edu

Andrea Brickey
Department of Mining Engineering, Colorado School of Mines, Golden, Colorado 80401, abrickey@mymail.mines.edu

Alexandra M. Newman
Division of Economics and Business, Colorado School of Mines, Golden, Colorado 80401, newman@mines.edu

Kelly Eurek
Strategic Energy Analysis Center, National Renewable Energy Lab, Golden, Colorado 80401, keurek@gmail.com

A classical problem in the mining industry for open-pit mines involves scheduling the production of notional three-dimensional production blocks, each containing a predetermined amount of ore and waste. That is, given operational resource constraints on extraction and processing, we seek a net present value-maximizing schedule of when, if ever, to extract each block in a deposit. We present a version of the problem, which some literature refers to as (*CPIT*). This constrained ultimate pit limit problem (i.e., open-pit production-scheduling problem variant) produces a sequence of blocks to extract given minimum and maximum bounds on production and processing capacity, and geospatial precedences. Our tutorial demonstrates methods to expedite solutions for instances of this model through variable definition, preprocessing, algorithmic choice, and the provision of an initial feasible solution. As such, our paper is relevant for any mining practitioner interested in production scheduling, and any operations researcher interested in a basic introduction before extending the boundaries of algorithmic development in this area.

Keywords: mine scheduling; mine planning; open-pit mining; surface mining; optimization; integer programming applications.
History: This paper was refereed.

Mine production scheduling is the process of determining an extraction sequence for notional three-dimensional blocks, each containing prespecified amounts of ore and waste. Large excavators and haul trucks extract and subsequently transport these blocks of material to a mill (i.e., processing plant) or to a waste site, depending on the expected profitability of the material. The rate at which the material is excavated and processed depends on initial capital expenditure decisions, permitting, contract agreements, projected sales, processing capacities, the presence of equipment (e.g., haul trucks and loaders), and the existence of infrastructure (e.g., roads and rail lines). Processed ore can be sold according to long-term contracts or on the spot market. Waste is left in piles, which must ultimately be remediated or reclaimed when the mine is closed.

Researchers have studied mine production scheduling since the late 1960s (Johnson 1968). Common variables in this integer-programming formulation include when, if ever, to extract a block to maximize the operation's net present value (NPV). Constraints include spatial sequencing between blocks and restrictions on available operational resources. However, hardware and software capabilities have only recently allowed solutions to realistic variants of the production-scheduling problem (Newman et al. 2010). Commercial software exists to address production scheduling in open-pit mines; under some circumstances it works well. Nonetheless, researchers have also shown that such software does not necessarily provide an optimal, or even a feasible, solution for realistic variants (De Kock 2007, Gaupp 2008, Somrit 2011). These shortcomings may stem from the fact that every mine possesses different attributes and policies. For example, some mines may stockpile (in a variety of ways) and some mines may invoke a policy whose decisions also include whether to send a block to a waste dump or to a processing plant. Our objective in this paper is to provide a tutorial for mining practitioners and applied operations researchers who are interested in building customized models for open-pit mine production scheduling and (or) who are interested in increasing the tractability of existing models. For a basic model, we provide fundamental insights often left unstated or even untested in the literature.

This paper is organized as follows: The *Model* section provides a description and mathematical formulation of the production scheduling model variant with which we work. In the *Solution Methodologies* section, we discuss methods for increasing the tractability of this model. Some of these methods are used, but are not well documented in the literature; others are novel. In *Results*, we give numerical results based on the application of these methodologies, and we conclude with a summary and directions for future research in *Conclusions*.

Model

This paper focuses on solving real-world instances of an open-pit production-scheduling model, also called the open-pit block-sequencing (OPBS) problem. At the strategic level, the ultimate pit limit (UPL) problem, based on the seminal work of Lerchs and Grossmann (1965), provides a solution for the ultimate pit limit contours of a deposit with the objective of maximizing total profit. This formulation includes geometry constraints; however, it does not factor in the effects of time, production, or blending constraints. At the tactical level, the OPBS problem includes the dimension of time, answering the question of when, if at all, to extract each block. Next, we show our formulations of the OPBS problem and methodologies to reduce their solution time. See Espinoza et al. (2013) for a discussion of the relationship between these two problems and their important attributes.

Three-dimensional block models used to determine the optimal block sequence can vary in size and block characteristics according to the deposit, or even the area within a deposit, being modeled. The geology, shape, and scale of a deposit can influence the block size, typically given by the dimensions of the block in x-, y-, and z-directions. For a narrow-vein gold deposit, the blocks may be oriented along strike with block size dependent on the width of the vein. For other deposits in which the mineral formation is highly dispersed, the height of the blocks may correspond to the scale of the equipment being utilized. In this tutorial, we depict the deposit of interest using a three-dimensional block model, where each block's characteristics are known with certainty.

Block models can include extensive data representing many physical properties of each block, including, but not limited to, their volume, tonnage, cost, metal recovery, and grade. In a deterministic model, such as the one we present in this paper, these data are assumed to be certain, enabling the a priori calculation of each block's value, and the calculation of operational resources (i.e., production and processing) required for its extraction and destination (e.g., a processing plant). The information stored in a block model are estimations based on the available data, such as drillhole samples, existing operations, and metallurgical testing. Metal recoveries represent the percentage of metal that can be recovered from processed ore. Ore grades reported in a block model can be in-situ grades, with a recovery percentage applied during revenue calculations, or as recovered grades with the recovery factor already applied. For this tutorial, we compute the revenue associated with a block as the product of the amount of metal recovered from the processing plant and an estimated sale price; we compute the operational cost as the product of the amount of ore and the mining and processing costs less the product of the amount of waste and the associated mining cost. If the difference between the revenue and the operational cost is negative, we classify the block as waste and compute its extraction cost as simply the product of the amount of waste and the associated mining cost. Otherwise, the block qualifies as an ore block. Based on this classification, we categorize each ore block as destined for the processing plant and each waste block as destined for the dump. Although this a priori categorization is generally regarded as a strong assumption in short-term scheduling models, it can be an appropriate simplification for strategic block-sequencing models.

These calculated block values, in turn, enable the generation and economic valuation of block extraction schedules, accounting for costs associated with different mining and processing methods, transportation modes, and revenues based on the expected market value of the mineral. However, depending on the size of the deposit and the block size, block models can contain millions of blocks, which precludes fast calculations of solutions to scheduling models. To improve the speed of these calculations, algorithms (typically, that of Lerchs-Grossman applied with an estimated metal price) can be used to determine the ultimate pit limits and to reduce model sizes by eliminating those blocks that fall outside of the limits. When conducting any model size reduction, we must ensure that all blocks that might be included in an optimal production schedule are retained. It is possible, for example, in the presence of lower bounds on operational resource constraints in the block extraction scheduling model, that certain techniques to reduce model size may exclude blocks that would otherwise be included in an optimal production schedule. Therefore, if the practitioner wishes to reduce the problem size by solving a series of (ultimate pit limit) problems, the authors recommend choosing higher economic (price) parameters, which would result in removing fewer blocks from the original model, and thus, a lower likelihood of removing a block that should be contained in the optimal solution to a block extraction schedule.

Aggregating adjacent blocks is a technique to further reduce model size. The decision to aggregate blocks from a model could be based on the location of blocks that share common characteristics (e.g., ore or waste content), deposit characteristics, the number of blocks in the model, the time fidelity in the formulation, and the required accuracy necessary in the resulting solution. Aggregation can be used when sufficient operational resources exist to extract many blocks in a given period, and must be used if the initial block size is not consistent with the mining method (e.g., the equipment is not capable of selectively mining at that block size). For our paper, we assume that the initial block size is suitable for the analysis. However, our techniques are valid for any large instance, regardless of whether that instance was arrived at by aggregating blocks.

Although mining only valuable ore blocks would be preferable, the order in which blocks can be mined is dictated by a maximum allowable slope angle of the deposit and by the blocks' relative positions to each other. The slope angle is determined by geological conditions, such as rock strength, and design parameters, such as roads. Typical geospatial precedence constraints between blocks in an open-pit mine require that the block immediately above the block in question be mined before the block in question can be mined (see Figure 1). The literature commonly contains more elaborate precedence constraints to preserve the necessary pit slope angles or adequate working areas for equipment. To retain the focus of this tutorial on optimization modeling, we address slope angles with the common, but simplified, "+" sign convention. Here, we assign precedences such that prior to mining a block b, the block b' directly above b, and those blocks adjacent to and on the same level as b' must first be mined (see Figure 2). Although these precedence constraints are an oversimplification

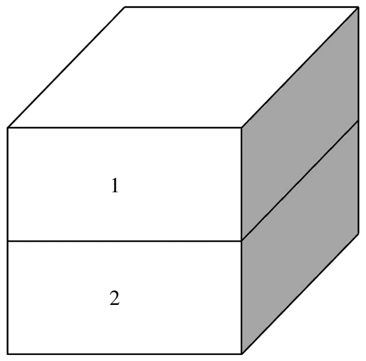

Figure 1: Block 1 must be mined prior to mining block 2.

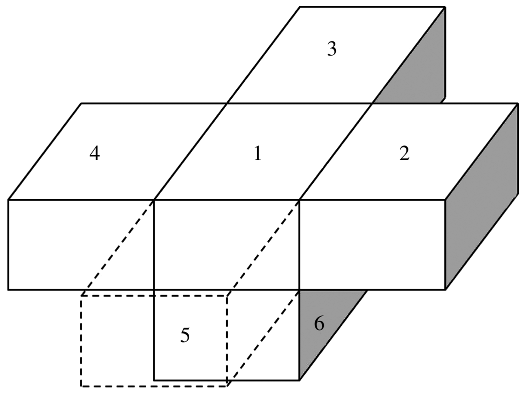

Figure 2: Blocks 1–5 must be mined prior to mining block 6.

of reality, they are commonly used in the academic literature; as of this writing, they are thought to be reasonable approximations of reality. For cases in which they are not reasonable approximations, the modeler may specify other precedence rules without any loss of generality. See Espinoza et al. (2013) for a discussion. The complete set of blocks on all levels above b, which must be mined prior to mining b, constitute b's complete predecessor set, or b's predecessor pit. A transitivity relationship exists between block precedences. By transitivity, we can say that a precedence relationship is immediate if no other pair of precedence relationships implies it. The subset of blocks within the complete predecessor set that possesses an immediate precedence relationship with b constitutes b's direct predecessor set.

It is important to ensure that the problem being solved is well defined. Different specifications of the open-pit mining problem exist, from the strategic ultimate pit limit problem, which identifies the most economical subset of blocks to extract, to the tactical precedence-constrained production-scheduling problem that identifies which blocks to extract and when and where to send them. In this paper, we focus on the deterministic, constrained pit limit problem (*CPIT*), as named in Chicoisne et al. (2012), which is a simplified version of the formulation presented in Johnson (1968). We use this variant, but generalize the problem in Chicoisne et al. (2012) by including nonzero minimum operational resource constraints, which require some extraction and (or) processing activities in each period. The solution to (*CPIT*) provides a NPV-maximizing block extraction sequence that satisfies minimum resource requirements and maximum resource capacities, in addition to the geospatial precedences. We also refer to (*CPIT*) as the monolith, in that it encompasses the entire formulation, as opposed to a formulation produced via a mathematical decomposition procedure, such as Lagrangian relaxation or Benders decomposition.

The mathematical formulation of the (*CPIT*) problem is a mixed-integer linear program. The objective is to determine the period in which each block has to be mined to maximize the NPV over a given time horizon. We represent the block mining precedence requirements and the resource availability requirements as constraints in the model. A detailed mathematical formulation of the problem is available in Appendix A.

Solution Methodologies

Reducing solution time for instances of the OPBS problem is imperative, because these can contain between 10,000 and millions of blocks, and between 10 and 100 periods, as Chicoisne et al. (2012) illustrate. Most commercial solvers, such as CPLEX (IBM 2011), utilize the branch-and-bound (B&B) algorithm to solve integer programs. Our solution methodologies are intended to enhance the solver's functionality by exploiting the special structure of the OPBS problem. The following solution methodologies attempt to reduce the solution time for instances of (*CPIT*) by (1) tightening the formulation; (2) solving the linear programming (LP) relaxation of the integer program at the root node with the proper algorithm and corresponding algorithmic settings; (3) reducing the problem size through variable elimination; and (4) providing the optimizer with an initial integer feasible solution. A reduction in solution time corresponds to a reduction in costs associated with in-house or consulting engineers who develop and analyze scheduling scenarios, and the reduction should contribute to an improved ability to solve larger-sized block models.

Improved Formulations

The formulation described in Appendix A can be improved by making the constraint set tighter and by replacing the set of decision variables that determine when a specific block must be mined, by a set of variables that indicate the period before which a block must be mined. We denote the formulation with the first and second improvements, ($CPIT^{\text{att}}$) and ($CPIT^{\text{by}}$), respectively. Appendix B provides a detailed description of the improved formulations.

Space precludes us from presenting all methods that improve our (*CPIT*) formulation; however, significant academic work exists based on adding valid inequalities (i.e., cuts). For a formulation with binary variables, such as ours, a valid inequality can assume the form of a packing constraint, explicitly precluding the simultaneous assignment of a value of 1 to a combination of variables, where such an assignment has been determined a priori to be infeasible. For example, if it is infeasible to extract both blocks b and b' in period t because of, for example, the resources required versus those available to extract the blocks, then the corresponding valid inequality is $w_{bt} + w_{b't} \leq 1$. Although numerous types of valid inequalities (e.g., covers, cliques) are available, a challenge for their generation lies in identifying valid (and helpful) infeasible variable combinations. Park and Park (1997) and Boland et al. (2012), respectively, identify techniques to generate valid cover and clique inequalities for precedence-constrained knapsack problems in general, whereas Gaupp (2008) presents empirical evidence of their benefit to the OPBS problem in particular.

Root Node Algorithm Performance

Attempting to solve large-scale integer programming problems using the incorrect LP algorithm at the root node can result in excessive run time before branching, whereas solving the same instance with the correct LP algorithm may yield a root node solution in a matter of seconds. The two problems (primal and dual), and three algorithms (primal simplex, dual simplex, and barrier), yield six problem-algorithm combinations. Testing these six combinations is worthwhile for practitioners who repeatedly solve instances using a single block model. Additionally, some optimizers, such as CPLEX (IBM 2011), include a network simplex algorithm that is able to more rapidly solve the LP when a significant portion of that LP's constraint matrix forms an underlying network structure. For an in-depth treatment of LP algorithmic selection, see Klotz and Newman (2013a).

Variable Elimination

The optimization model need not include variables that must assume a value of 0 in any feasible solution; therefore, the act of identifying and eliminating such variables a priori results in a smaller model. In the OPBS problem, the variables y_{bt} indicate whether to extract block b at period t. Periods t', in which it is infeasible to extract block b, imply that the variable $y_{bt'}$ must assume the value of 0 in any feasible solution, allowing us to then exclude $y_{bt'}$ from the formulation. For example, consider the simple two-block model, which Figure 1 depicts for two periods, with production sufficient to extract only one block per period. Because block 1 must be extracted prior to block 2, it is infeasible in mathematical terms (i.e., impossible in practice) to extract block 2 in period 1. Therefore, the variable value of y_{21} must equal 0 in any feasible solution; thus, including y_{21} in the model is unnecessary.

For each block b, there exists a period $\underline{t}$ before which it is not possible to extract b while satisfying all constraints in periods $\{1 \ldots \underline{t}\}$. We refer to this period $\underline{t}$ as block b's true early start time (TES_b). Calculating TES_b is not trivial, because it requires identifying the earliest period by which all blocks b'' in b's predecessor pit $\mathcal{B}_b$ may be extracted while satisfying all constraints in each period; performing this identification could require solving an integer program for each block b. Although identifying TES_b for each block b may not be practical, we can still determine periods before which it is impossible to extract b, and thereby identify variables to eliminate from the formulation. This is possible by comparing the aggregate material in the predecessor pit $\mathcal{B}_b$ with the aggregate operational resource requirements and capacities for multiple periods (e.g., the aggregate capacity for t' periods is given by the sum of the capacities in periods $\{1..t'\}$). Next, for a given block b, we review early starts (ES_b), before which b may not be extracted because of maximum resource capacities, and develop enhanced early starts (EES_b), before which b may not be extracted because of both maximum resource capacities and minimum resource requirements. The value of TES_b may be later than these early starts; however, it cannot be earlier. Equation (1) shows the relationship between these three periods before which block b may be extracted:

$$ES_b \leq EES_b \leq TES_b. \tag{1}$$

There also exists analogous late starts (LS_b) (Gaupp 2008), in which variable values are fixed to 1 in ($CPIT^{\text{by}}$) and eliminated in ($CPIT^{\text{at}}$), corresponding to those block-period combinations for which block b must already have been extracted. Late starts exploit the fact that, as a result of precedence constraints, if block b is included in the predecessor pit of another block $\hat{b}$, then $\hat{b}$ may not be extracted before b is extracted. This, combined with the minimum resource requirements, forces block b to be extracted no later than some period LS_b. We find that although helpful for improving tractability in some instances, late start times do not generally reduce (via fixing or elimination) a sufficient number of variables, such that their inclusion has a significant performance impact.

Variable Elimination: Early Starts (ES). One necessary, but not sufficient, condition for block b to be extracted by $\underline{t}$ is that all of the blocks b'' in b's predecessor pit $\mathcal{B}_b$ be extracted no later than period $\underline{t}$. The earliest possible time any block b may be extracted is after all blocks b'' in b's predecessor pit $\mathcal{B}_b$ have been extracted, while operating at maximum production capacity. Once extracted, the earliest possible time an ore block $\check{b}$ may be processed is after all ore blocks $\check{b}''$ in $\check{b}$'s predecessor pit $\mathcal{B}_b$ have been processed, while operating at maximum processing capacity. Because (*CPIT*) does not consider stockpiling, a block may not be extracted prior to the more conservative (later) of the earliest times in which the block may be extracted or processed. More generally, a block b may not be extracted prior to the latest period in which the most constraining operational resource may handle all of b's predecessors. We call this earliest extraction time based on maximum resource capacities the block's early start (*ES*). Gaupp (2008), Amaya et al. (2009), and Chicoisne et al. (2012) employ the technique of using early start times to reduce variables in open-pit models. We present notation, used in the text in the following sections, and an algorithm for determining a block's early start time in Appendix C.

Variable Elimination: Enhanced Early Starts (EES). Although the incorporation of minimum operational resource constraints further complicates the problem of finding an optimal solution, it also affords opportunities for eliminating additional variables. In general, imposing additional constraints renders more variable values infeasible, and by identifying those variables whose value must equal 0 in any feasible solution, we may further reduce the problem's size.

From Gaupp (2008), the period ES_b is determined by the maximum resource capacities, which limit the rate at which the blocks b'' in $\mathcal{B}_b$ may be extracted and processed. Block b may not be extracted earlier than ES_b; however, it may also be true that block b may not be extracted earlier than some period later than ES_b, if block b's predecessor set is not feasible regarding the minimum resource requirements. In this sense, ES_b, as calculated previously, could be loose and may be tightened by accounting for additional constraints. Therefore, we define the enhanced early start time (EES_b) as that period before which it is infeasible to extract block b based on both the minimum resource requirement and maximum resource capacity constraints.

Consider the determination of block 7's early start time in the eight-block example shown in Figure 3. All blocks weigh 10 tons, blocks 6 and 8 are ore blocks, and lower-level blocks' predecessors include the three overlying blocks (e.g., block 6's predecessors are blocks 1, 2, and 3). For the case of a maximum production capacity of 40 tons per period, and no minimum processing requirement (i.e., $\bar{e}_1 = 40$, $\underline{e}_2 = 0$), block 7 may be extracted during period $t = 1$ using an extraction sequence of 2-3-4-7. Hence, in this case, block 7 has an early start time of $ES_7 = 1$.

For the case in which there is an additional minimum processing requirement of 10 tons of ore tonnage per period ($\bar{e}_1 = 40, \underline{e}_2 = 10$), the one-period extraction sequence of 2-3-4-7 does not contain an ore block to process in $t = 1$, and is therefore infeasible. A two-period feasible extraction sequence of 1-2-3-6 in $t = 1$ and 4-7-5-8 in $t = 2$ provides total tonnage no greater than $\bar{e}_1$, and ore tonnage no less than $\underline{e}_2$, in both periods. Therefore, extracting 7 in $t = 1$ satisfies the maximum production constraint, yielding $ES_7 = 1$, but extracting block 7 prior to $t = 2$ violates the minimum processing constraint, yielding $EES_7 = 2$. Because $EES_7 = 2$ is greater than $ES_7 = 1$, we may eliminate additional variables, excluding w_{71}

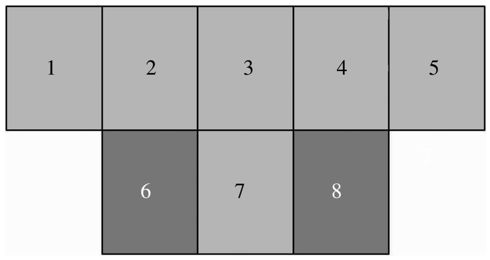

Figure 3: In this cross-sectional view of an eight-block pit, blocks 6 and 8 are ore blocks, and precedences follow the "+" sign convention in two dimensions, similar to that depicted in Figure 2. For example, a feasible schedule must call for extracting blocks 1, 2, and 3 prior to extracting block 6.

and y_{71} from the ($CPIT^{by}$) and ($CPIT^{at}$) formulations, respectively.

The preceding *EES* discussion focuses only on the minimum processing requirement, because satisfying the minimum production requirement is always possible. This follows because extracting any block, ore or waste, from anywhere in the pit, including surface blocks unconstrained by precedence requirements, contributes to satisfying minimum production. Therefore, for a case in which minimum processing is binding in the ES_b^{th} period (i.e., the period corresponding to ES_b) and minimum production is not satisfied, extracting any blocks external to $\mathcal{B}_b$ can be used to increase total tonnage and satisfy minimum production during the ES_b^{th} period.

Unlike in the small preceding example, we cannot generally determine block b's EES_b by inspection. In Appendix D, we present both an algorithm for determining a block's enhanced early start time and associated notation, which is also used next. A positive residual processing requirement ($\hat{e}_{2b}$) in the ES_b^{th} period after extracting $\mathcal{B}_b$ (i.e., $\hat{e}_{2b} = \underline{e}_2 \cdot ES_b - S_{2b} > 0$) indicates that $EES_b \geq ES_b$. However, eliminating the variable w_{b,ES_b} (or y_{b,ES_b}) requires proving that $EES_b > ES_b$, which is not always trivial. Specifically, determining EES_b can be complicated for the situation in which $\hat{e}_{2b} > 0$, and there also exists a positive residual production capacity ($\hat{e}_{1b}$) in the ES_b^{th} period, (i.e., $\hat{e}_{1b} = \bar{e}_1 \cdot ES_b - S_{1b} > 0$).

For the case in which $\hat{e}_{1b} \geq \hat{e}_{2b}$, it may still be possible to extract sufficient ore blocks outside of $\mathcal{B}_b$ to satisfy $\underline{e}_2$ in the ES_b^{th} period. Adequate residual production capacity exists to extract the ore blocks necessary to satisfy the residual processing requirement, if those ore blocks are accessible and can be extracted in a feasible sequence. Therefore, showing $EES_b > ES_b$ generally requires proving that there does not exist an extraction sequence of blocks external to $\mathcal{B}_b$ containing sufficient ore blocks to meet or exceed $\hat{e}_{2b}$, while not exceeding $\hat{e}_{1b}$. In the worst case, this is an exponentially complex problem of enumerating each possible block sequence, likely requiring the solution of an integer program. Therefore, for the case in which $\hat{e}_{1b} \geq \hat{e}_{2b}$, without solving an integer program, we must make the conservative assignment of $EES_b = ES_b$, and are unable to eliminate the variable w_{b,ES_b} (or y_{b,ES_b}).

For the case in which $\hat{e}_{1b} < \hat{e}_{2b}$, even if sufficient ore blocks are available, it is impossible to satisfy $\underline{e}_2$ in the ES_b^{th} period without violating $\bar{e}_1$, guaranteeing that $EES_b > ES_b$. Although EES_b may exceed ES_b by more than one period, establishing this requires proving the nonexistence of feasible extraction sequences in successive periods. Therefore, in this situation, $EES_b > ES_b$, but we cannot prove by how much; therefore, we make the conservative assignment $EES_b = ES_b + 1$, and eliminate the variable w_{b,ES_b} (or y_{b,ES_b}).

Again, it is possible to relax the assumption of time-invariant maximum resource capacities, although we do not include the corresponding more general algorithm here. However, a formula analogous to Equation (C1) would require interaction between different resources (e.g., r and r'). This complicates a generic formula, because it would then be highly dependent on the resources to which r and r' refer; in our case, r and r' have a very specific relationship, not a general relationship. Our formulation contains resource r (i.e., production) with a nonzero value for all blocks and resource r' (i.e., processing) with a nonzero value for some of the blocks. And, by comparing r and r' in a particular way (i.e., the lower bound on r' with the upper bound on r) we can, under certain circumstances, increase the start time of a block by one period.

Initial Integer Feasible Solution (IIFS)

Providing the optimizer with an IIFS should expedite the B&B search for an optimal solution both by allowing an initial lower bound to preclude evaluation of dominated solutions (i.e., solutions not guaranteed to be feasible, but if feasible, are guaranteed to have an objective function value lower than that achieved with the IIFS), and by assisting the optimizer to more effectively tailor its search strategies. The challenge is identifying such a solution. Some heuristics to this end are as follows:

- Greedy algorithms can quickly identify an initial solution to a problem specification in which resources are only constrained by maximum capacities (i.e., no minimum requirements exist). For example, we may order blocks, descending by profit, and then choose to extract each block, along with its predecessor set, if sufficient resource capacity exists. We repeat this procedure for each period, creating a block extraction schedule for the entire time horizon.

• Chicoisne et al. (2012) and Moreno et al. (2010) extend this simple approach by exploiting the precedence constraints' underlying network structure to induce a topological sorting of nodes (i.e., blocks). This sorting may be based on any attribute, including LP relaxation values, as demonstrated by their *Expected-Time TopoSort* heuristic. Using these techniques, Moreno et al. (2010) show that initial feasible solutions may be obtained quickly for large data sets.

• Although greedy heuristics work well when the problem specification includes only maximum resource capacities, finding a feasible solution is significantly complicated by including minimum resource requirements. A sliding time-window heuristic (STWH) (Pochet and Wolsey 2006) may be more likely to determine an integer feasible solution than a simple greedy approach. With the STWH, a subproblem is formed by partitioning the set of periods in the horizon T into windows (i.e., $T \equiv T^{\text{Fix}} \cup T^{\text{Cont}} \cup T^{\text{Int}}$), within which variable values are fixed (T^{Fix}), or their integrality is either relaxed (T^{Cont}) or enforced (T^{Int}). Solving this subproblem produces a partial solution, which is integer feasible for variables in T^{Int}. Then, a subsequent subproblem is formed by changing the partition of T (i.e., the window slides), so integrality is enforced in a separate set of periods, and the procedure is repeated. Finally, linking the partial integer feasible solutions from each subproblem creates a complete solution. Although this approach does not guarantee an integer feasible solution for a monolith with minimum resource requirements, even an infeasible solution may assist the optimizer in its search for an optimal solution. See Cullenbine et al. (2011) for further discussion.

• An alternative approach leverages the seminal work of Lerchs and Grossmann (1965) by finding, within the entire pit, a maximum-value subpit containing sufficient ore and material to fulfill the total resource requirements over the problem's time horizon. Then, an integer program may identify an integer feasible sequence within that subpit, which satisfies the resource constraints in each period. This resulting solution, combined with setting variable values to 0 for all blocks outside the subpit, forms a complete initial feasible solution, as Lambert and Newman (2013) fully explain.

Results

We empirically test the strategies outlined in the previous section using 12 open-pit mine block models derived from two master data sets. The first master data set is an open-pit mine block model consisting of 10,819 blocks, approximately 5,600 tons each, after the elimination of irrelevant blocks (e.g., waste blocks at the bottom level of the pit, and air blocks of 0 tonnage used for graphical display). From this master data set, referred to as 10k, we perturb each block's mineral content by ±5 percent to generate seven more block models, identified as $10kA, 10kB, \ldots, 10kG$; we collectively refer to them as the 10k data sets.

The second master data set is the well-known Marvin test data set available from Whittle software, which Norm Hanson created while at the Royal Melbourne Institute of Technology (Hanson 2012). This fictional, yet realistic, data set models typical deposits found in New South Wales, Australia, and contains 53,668 blocks, approximately 62,400 tons each, after eliminating air blocks from the top of the pit. We use four smaller subsets from this full Marvin for test cases, two each of size 18,300 (18kA and 18kB), and 25,620 (25kA and 25kB) blocks. We refer to these collectively as the Mrv data sets. Table 1 displays summary characteristics of all data sets.

	Number of blocks			Tonnage (Mtons)	
Data set	Total	Ore	Waste	$\sum_{b\in B} a_{1b}$	$\sum_{b\in B} a_{2b}$
10k	10,819	1,423	9,396	60.57	7.92
10kA	10,819	1,423	9,396	60.57	7.92
10kB	10,819	1,414	9,405	60.57	7.87
10kC	10,819	1,414	9,405	60.57	7.87
10kD	10,819	1,410	9,409	60.57	7.85
10kE	10,819	1,418	9,401	60.57	7.89
10kF	10,819	1,423	9,396	60.57	7.92
10kG	10,819	1,420	9,399	60.57	7.90
10k avg	10,819	1,418	9,401	60.57	7.89
18kA	18,300	2,032	16,268	1,145.47	146.62
18kB	18,300	1,436	16,864	1,137.42	103.39
25kA	25,620	2,248	23,372	1,595.62	162.03
25kB	25,620	2,805	22,815	1,601.13	199.95
Mrv avg	21,960	2,130	19,830	1,369.91	153.00

Table 1: We test our solution strategies on these 12 data sets with characteristics including the number of blocks by type (e.g., total, ore, waste), and the tonnage required to extract all blocks and process all ore blocks.

We formulate our models in the AMPL programming language, version 11.11 (AMPL 2009) and then pass this formulation to the CPLEX solver, version 12.2.0, (IBM 2011), which runs on a Sun X4150, with 2×2.83 GHz processors, and 16 GB RAM.

Root Node Algorithm Performance

The algorithm selection for solving the LP relaxation of the problem instance at the root node can dramatically impact overall solution time. Consider the results displayed in Table 2 from solving the LP relaxations with a horizon of $\tau = 10$. For the 10*k* data sets, solving with CPLEX default parameter settings (i.e., dual simplex performed on the dual problem as in IBM 2011) finds no feasible solution within 1,800 seconds for any of the instances tested, whereas solving the primal problem using the primal simplex method finds an optimal solution within 1,800 seconds for six of the eight instances. The barrier algorithm generally performs best on the 10*k* sets, finding the optimal solution in less than 900 seconds. The Mrv instances exhibit the best performance using the primal simplex method, whereas the barrier algorithm is unable to find an optimal solution within 1,800 seconds. A common approach that gives good performance across all of our instances suggests CPLEX's network simplex algorithm (Netopt), which is able to extract and exploit the problem's underlying network structure.

		(*P*)—Primal problem			(*D*)—Dual problem			
Data set	CPLEX default	*PS**	*DS**	Barrier	*PS*	*DS*	Barrier	Netopt
10*k*	†[b]	998[a]	‡[a]	807	†[a]	†[a]	831	992
10*kA*	†[b]	1,606[a]	‡[a]	732	†[a]	†[a]	740	1,587
10*kB*	†[b]	1,298[a]	‡[a]	707	†[a]	†[a]	737	1,291
10*kC*	†[b]	1,478[a]	‡[a]	712	†[a]	†[a]	805	1,470
10*kD*	†[b]	1,093[a]	‡[a]	733	†[a]	†[a]	852	1,075
10*kE*	†[b]	†[a]	‡[a]	732	†[a]	†[a]	743	1,976
10*kF*	†[b]	1,464[a]	‡[a]	730	†[a]	†	691	1,451
10*kG*	†[b]	†[a]	‡[a]	684	†[a]	†	758	2,036
18*kA*	352[b]	105	1,380[a]	†	†[a]	†[a]	†	85
18*kB*	210[b]	86	1,374[a]	‡	†[a]	†[a]	†	66
25*kA*	‡[b]	135	1,086[a]	‡	†[a]	†	†	107
25*kB*	‡[b]	604	1,008[a]	‡	†[a]	†	‡	577

Table 2: When solving problem instances for $\tau = 10$ years with no *ES* variable reductions, we obtain varying LP relaxation solution times (in seconds) by using different problem-algorithm combinations.

[a]**CPLEX introduces a perturbation, suggesting degeneracy.**

[b]**CPLEX reports default settings result in solution being found with dual simplex on the dual problem; however, CPLEX uses different algorithms on multiple threads during concurrent optimization.**

†**Time limit of 1,800 secs in phase I.**

‡**Time limit of 1,800 secs in phase II.**

***PS: primal simplex; DS: dual simplex.**

The difference in performance between the primal simplex method and barrier algorithm on the Mrv and 10*k* data sets can be explained as follows: using the primal simplex on the latter data sets requires a significant amount of solution time in phase I, has difficulty because of degeneracy, and fails to utilize a favorable pricing scheme until a significant number of phase II iterations have passed. By contrast, the same algorithm on the Mrv data sets spends little time in phase I, exhibits no degeneracy, and recognizes a good pricing scheme (i.e., devex) upon switching to phase II. Using the barrier algorithm on the 10*k* data sets requires little time in crossover (i.e., finding a basic solution from one lying on the boundary), whereas the Mrv data sets require a long time for this procedure. Arguably, alternate parameter settings (e.g., those to address degeneracy up front, or those to diminish the time spent in crossover) might help mitigate the performance discrepancies between the two algorithms applied to these two data sets. However, experimentation at this level was beyond the scope of our numerical testing.

"By" vs. "At" Formulation

In solving integer programs, the tighter the bound (i.e., LP relaxation objective function value), the better the B&B algorithm performance (Bertsimas and Weismantel 2005, p. 11). In Table 3, we present LP relaxation objective function values (z_{LP}) to quantify the benefit of tighter formulations. The second and third columns contain LP relaxation values from the *at* formulation with no variable reductions, and with our greatest variable reductions from enhanced early starts, respectively. Although this preprocessing does tighten the bound, using either the *att* or *by* formulation (because they provide the same LP relaxation objective function value, and as such, are equivalently strong), in conjunction with these reductions, tightens the bound to a much greater extent. As such, we achieve all further results using the *by* (equivalently, *att*) formulation. We present the percentage

	Linear program relaxation objective function value ($ M)			
Data set	$z_{LP}(CPIT^{at})$	$z_{LP}^{EES}(CPIT^{at})$	$z_{LP}^{EES}(CPIT^{by})$[1]	ζ(at, by) (%)
10*k*	25.80	18.84	10.97	41.8
10*kA*	25.93	18.87	11.01	41.7
10*kB*	25.92	18.90	10.98	41.9
10*kC*	25.80	18.77	10.91	41.9
10*kD*	25.79	18.76	10.96	41.6
10*kE*	25.71	18.77	10.90	42.0
10*kF*	25.92	18.95	11.01	41.9
10*kG*	25.77	18.83	10.96	41.8
10*k* avg	25.83	18.84	10.96	41.8
18*kA*	146.57	146.19	133.08	9.0
18*kB*	117.33	117.33	114.26	2.6
25*kA*	143.15	143.15	140.78	1.7
25*kB*	132.69	123.82	85.70	30.8
Mrv avg	134.94	132.62	118.46	11.0

Table 3: When solving problem instances for $\tau = 10$ years, using both variable reductions (*EES*) and the *by* formulation provide tighter LP relaxation values than solving the same problem instances with no variable reductions and the *at* formulation.

[1]$z_{LP}^{EES}(CPIT^{att})$ **possesses the same objective function value as** $z_{LP}^{EES}(CPIT^{by})$.

reduction in the LP relaxation value, which quantifies the relative strength of $(CPIT^{by})$ over $(CPIT^{at})$ as:

$$\zeta(\text{at, by}) = 100\% \cdot \frac{z_{LP}^{EES}(CPIT^{at}) - z_{LP}^{EES}(CPIT^{by})}{z_{LP}^{EES}(CPIT^{at})}. \quad (2)$$

Variable Elimination

Our data sets each contain a significant number of variable reductions attributable to *ES* and *EES*, as Table 4 shows. Although instances with fewer variables and constraints generally lead to faster solution times than instances with no variable reductions (*NoES*), occasionally CPLEX's proprietary heuristics are able to find solutions more quickly than would be suggested by theory (e.g., in the cases of 10*kF* and 18*kB*). However, Table 4 demonstrates an additional savings of 17 percent in solution time, on average, for the 10*k* data sets beyond what the early start reductions alone can achieve. We find the number of variable values fixed by late starts, introduced by Gaupp (2008), does not justify the effort required for their calculations. For example, in the 10*k* master data

	Number of variables			Solution time (secs)[1]			Solution time reduction (%)	
Data set	*NoES*[2]	*ES*	*EES*	*NoES*	*ES*	*EES*	$(NoES - ES)/NoES$	$(ES - EES)/ES$
10*k*	108,190	42,185	40,373	16,876	4,886	3,678	71%	25%
10*kA*	108,190	42,185	40,362	17,026	7,072	3,502	58%	50%
10*kB*	108,190	42,185	40,366	29,059	3,897	3,538	87%	9%
10*kC*	108,190	42,185	40,379	21,535	4,191	3,529	81%	16%
10*kD*	108,190	42,185	40,353	16,750	4,714	4,278	72%	9%
10*kE*	108,190	42,185	40,371	24,024	6,256	3,983	74%	36%
10*kF*	108,190	42,185	40,380	18,044	3,976	5,814	78%	−46%
10*kG*	108,190	42,185	40,359	16,356	4,377	4,223	73%	4%
10*k* avg	108,190	42,185	40,368	19,960	4,921	4,068	74%	17%
18*kA*	183,000	178,711	172,113	‡	‡	35,449	*	*
18*kB*	183,000	178,818	172,129	18,973[3]	27,399[4]	26,553	−44%	3%
25*kA*	256,200	229,808	216,586	‡	‡	‡	*	*
25*kB*	256,200	230,013	216,634	‡	‡	‡	*	*

Table 4: When solving problem instances for $\tau = 10$ years within a time limit of 10 hours, IP solution times are generally shorter with a greater number of variable reductions.

[1]**Solution times do not include the time required to calculate *ES*s and *EES*s. We generate both *ES*s and *EES*s for all blocks in a given data set in fewer than 10 seconds.**

[2]***NoES*** **indicates no variable reductions.**

[3]**Heuristic found first and final integer solution after 18,844 seconds.**

[4]**Heuristic found first and final integer solution after 27,350 seconds.**

‡**Time limit (36,000 secs) reached with no integer solution.**

***Unable to calculate due to lack of solution time values.**

set, only 48 of the 108,190 variable values may be fixed because of late starts. Hence, none of our runs include variables fixed to their *LS* values. The relevant CPLEX parameter settings for all runs include *primal* (i.e., set up the primal formulation), *predual* -1 (i.e., solve the primal problem), *startalgorithm* 3 (i.e., use the network simplex algorithm to solve the LP relaxation at the root node), *subalgorithm* 1 (i.e., use the primal simplex algorithm to solve the linear programs at all nodes of the B&B tree, other than at the root node), *mipemphasis* 1 (i.e., emphasize feasibility over optimality when searching through the B&B tree), *varsel* 4 (i.e., branch in the tree based on pseudo-reduced costs), *probe* -1 (i.e., preclude spending time to explore binary variable assignment implications, and then preset variables accordingly), *mipgap* 0.02 (i.e., solve the problem to within two percent of optimality); the meanings of these parameter settings are specified in IBM (2011).

Initial Integer Feasible Solution (IIFS)

Providing the optimizer with an IIFS expedites the B&B algorithm by precluding the optimizer from evaluating dominated solutions and by enhancing the optimizer's search path (Klotz and Newman 2013b). Additionally, the optimizer can leverage aggressive cut generation to tighten the upper bound and exploit the use of its own heuristic more frequently to search for better integer solutions. Our technique requires two phases; in the first, we generate the IIFS with the STWH; in the second, we solve the monolith with that IIFS. Deducting the total time required for these two phases from that required to solve the monolith without an IIFS yields the reduction in solution time attributable to the use of an IIFS.

Implementing the STWH requires selecting the number of windows, and in each of those windows, how many periods in which to enforce integrality and to fix variable values. Enforcing integrality in fewer periods requires more windows of relatively easier integer programs to solve. Conversely, enforcing integrality in more periods requires fewer windows of relatively more difficult integer programs to solve. We find the best STWH solution times for $\tau = 10$ result from implementing five windows, fixing and enforcing integrality in two periods (i.e., $|T^{\text{Fix}}| = |T^{\text{Int}}| = 2$) for each window.

The results in the objective function value columns of Table 5 show that the STWH generates good quality solutions. The solution time columns of Table 5

	Objective function value ($ M)		Solution time (secs)		
Data set	z_{LP}^{EES}	z_{STWH}^{EES}	Mono with no IIFS	STWH + Mono with IIFS	IIFS % reduction
10*k*	10.97	10.64	3,678	2,323	37%
10*kA*	11.01	10.65	3,502	2,523	28%
10*kB*	10.98	10.58	3,538	2,595	27%
10*kC*	10.91	10.27	3,529	2,145	39%
10*kD*	10.96	10.72	4,278	924[1]	78%
10*kE*	10.90	10.48	3,983	2,447	39%
10*kF*	11.01	10.54	5,814	3,430	41%
10*kG*	10.96	10.61	4,223	2,285	46%
10*k* avg	10.96	10.56	4,068	2,334	42%
18*kA*	133.08	125.20	35,449	24,256	32%
18*kB*	114.26	112.73	26,553	1,765[1]	93%
25*kA*	140.78	†	‡	†	*
25*kB*	85.70	†	‡	†	*
Mrv avg[2]	123.67	118.96	31,001	13,011	62%

Table 5: We see improvements in both solution quality and time (secs) for the monolith when provided with an IIFS from the STWH for $\tau = 10$, and solved to within a mipgap tolerance of two percent.
[1]STWH generated initial solution within 2% mipgap for monolith.
[2]Only includes completed runs for Mrv data sets.
†Time limit (36,000 secs) reached during STWH with no integer solution.
‡Time limit (36,000 secs) reached with no integer solution.
*Unable to calculate as a result of a lack of solution time values.

show that generating the IIFS and then solving the monolith with that IIFS reduces solution time relative to solving the monolith outright. These reductions in solution time are 42 and 62 percent, on average, for our 10k and Mrv data sets, respectively.

Conclusions

Practitioners face numerous challenges when implementing mathematical programs to solve difficult real-world problems, such as the OPBS problem. This paper demonstrates a variety of techniques that expedite solutions: (1) alternative variable definitions enabling a tighter formulation, (2) algorithmic selection for solving the root node linear programming relaxation, (3) variable elimination, and (4) providing the solver with an initial feasible solution. Although we show that all techniques can be effective, the success of their implementation can be highly dependent on the specific problem instance (e.g., data set characteristics, time horizon). For example, recall from Table 2 that when solving the root node LP relaxation for the 10k data sets, the barrier algorithm performs best, whereas for the Mrv data sets, primal simplex performs best. Also, although fixing variable values via late starts did not justify their calculation for our data sets, they may provide justifiable benefits for problem instances with tight lower bounds on resource constraints or for instances that represent mines with different topologies.

Theory suggests that the aforementioned techniques should reduce solution time; however, this is not always the case when using a modeling language and commercial optimizer. Next, we describe some of the more frustrating, and sometimes surprising, difficulties we encountered in our implementation.

Optimizer Presolve and Heuristics. Commercial optimization software may improve overall performance with various presolve activities (e.g., problem size reduction, generation of the equivalent dual, validation of mixed-integer programming (MIP) starts), and with heuristics that modify search strategies. Presolve can provide significant improvements in performance; however, in doing so, the problem may be modified in ways the user might not prefer. For example, one method to reduce time spent in solving the LP relaxation of a MIP's root node is to pass the optimizer an optimal basis, previously determined when finding an IIFS. However, to do so requires turning off the MIP presolve to ensure no basic variables are eliminated via, for example, CPLEX's probing feature, which fixes deduced variable values, prior to solving the root relaxation. Hence, the user saves time otherwise spent solving the LP at the root node, but loses time savings that might have been achieved during B&B.

Heuristics (see Achterberg 2007), can also significantly improve performance. However, by definition, their application does not constitute an exact approach, and as such, provides no guarantee of performance. Introducing a technique that theoretically should provide superior performance does not always result in reduced solution time, because a heuristic implemented within the solver's B&B algorithm may get lucky and find a solution more quickly (e.g., see the 10kF results in Table 4). Still, superior formulations, problem reduction techniques, and providing the solver with an initial solution all result in a simpler problem to solve; therefore, they should generally yield faster solution times.

Optimizer Parameter Settings. Commercial solvers afford the user some control, via the use of algorithmic parameter settings, over how an algorithm is executed. Although these user-adjusted parameters can impact solution time, each parameter's individual effectiveness is not always explicitly clear. Internal heuristics, influenced directly by the parameter settings and only indirectly by the user, determine the actual search path, which has a large impact on solution time. Therefore, modifying any one parameter setting, in conjunction with other parameter settings, may alter the heuristic's search path, resulting in a significant change in solution time. This indirect, interactive nature of the numerous parameters, each with their various settings, can make it difficult to find the single combination providing the best performance (i.e., the fastest solution time).

CPLEX includes a tuner tool (IBM 2011) to help identify that best combination of parameter settings. The tuner, however, can recommend very different settings for slight problem variations (i.e., the same data set with different time horizons, or slightly different data sets, such as 10kA and 10kC, with the same time horizon). These different recommended parameter settings are not just artifacts of the tuner, but result from the varying search paths previously mentioned. A reasonable approach is to obtain initial settings from the tuner and then test modified parameter settings based on the user's knowledge of the problem. For example, tuning for some of the $10k$ data sets suggests precluding initial probing and using pseudo-reduced costs to select the subnode variable on which to branch (CPLEX *probe* -1 and *varsel* 4, respectively). Although these perform better than CPLEX default settings, even faster solution times result from adding a search strategy of emphasizing feasibility over optimality (CPLEX *mipemphasis* 1). This follows from the inherent difficulty of finding feasible solutions to the OPBS problem. To select a consistent group of settings to employ when solving all models, the user must sometimes accept significant performance trade-offs, such as those that Table 2 discusses.

The aforementioned lack of direct user control over the solver suggests that reductions in solution time may not all be the result of our techniques. Although theory supports our results, it is possible that our techniques happen to provide (this magnitude of) reductions because of the solver's internal algorithms and heuristics. A valid avenue for future research is to implement the techniques described in this paper with other solvers to verify similar solution-time reductions.

Solving Linear Programs. As discussed in the *Root Node Algorithm Performance* section, effectively solving LPs may significantly reduce solution times. For example, the number of LPs solved in the monolith alone versus with an IIFS provided by the STWH differs significantly, on average. Solving the monolith requires solving one root relaxation, whereas solving the monolith with an IIFS (via our variant of the STWH) requires solving the monolith's root relaxation, in addition to the five root relaxations required for the STWH (one for each window). In the STWH, the cumulative time required to solve the root relaxation LPs can be greater than that required to execute the B&B algorithm. For example, in one instance ($10kG$), the STWH solves in 1,940 seconds, but only executes B&B for 40 seconds. This means 96 percent of the time is spent performing presolve, executing heuristics, and solving LPs. We find it helpful to approach solving the OPBS problem by explicitly recognizing the separate and very significant challenge to solving these LPs.

Despite performance irregularities and the intractability of many large detailed models that use current state-of-the-art hardware and software, computing power has made great strides in the past several decades (Bixby and Rothberg 2007). Knowing how to carefully formulate models and appropriately invoke solvers can transform an intractable instance into a solvable one. In this tutorial, we concentrate on (*CPIT*), a specific variant of the OPBS problem. The reader should be able to glean computational insights not only for this problem variant, but for other integer programming mine production scheduling problems.

Acknowledgments
We wish to acknowledge Dr. Ed Klotz of IBM for his patience, expertise, and willing assistance regarding explanations of the interaction between CPLEX parameters, interpretations of node logs, and clarification on the subtleties involved with solving the root relaxation of an integer program with CPLEX.

Appendix A. Mathematical Formulation of the Constrained Pit Limit Problem (*CPIT*)

Indices and sets:

- $b \in B$: set of all blocks b,
- $b' \in B_b$: set of blocks that must be extracted directly before block b, (i.e., b's direct predecessors),
- $b'' \in \mathscr{B}_b$: set of all blocks which must be extracted before block b, (i.e., b's predecessor pit: $B_b \subseteq \mathscr{B}_b$),
- $t \in T$: set of periods t,
- $r \in R$: set of operational resources r ($1 =$ production of ore and waste, $2 =$ processing of ore).

Data:

- τ: length of time horizon (i.e., $|T| \equiv \tau$),
- v_{bt}: NPV generated by extracting block b in period t (\$),
- a_{rb}: nonnegative amount of operational resource r used to process block b (tons),
- $\underline{e}_r/\bar{e}_r$: nonnegative, per-period minimum required usage/maximum usage capacity for operational resource r (tons).

Decision Variables:

$$y_{bt} = \begin{cases} 1 & \text{if block } b \text{ is extracted at time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{A1}$$

Objective Function:

$$(CPIT) \quad \max \sum_{b \in B} \sum_{t \in T} v_{bt} y_{bt} \tag{A2}$$

Constraints:

$$y_{bt} \le \sum_{t' \le t} y_{b't'} \quad \forall\, b \in B,\; b' \in B_b,\; t \in T, \tag{A3}$$

$$\underline{e}_r \le \sum_{b \in B} a_{rb} y_{bt} \le \bar{e}_r \quad \forall\, r \in R,\; t \in T, \tag{A4}$$

$$\sum_{t \in T} y_{bt} \le 1 \quad \forall\, b \in B, \tag{A5}$$

$$y_{bt} \in \{0, 1\} \quad \forall\, b \in B,\; t \in T. \tag{A6}$$

The objective function (A2) seeks to maximize the NPV of all extracted blocks. Although intuitive, our maximization of NPV is not a formulation requirement, and without loss of generality, we could consider other objective functions provided they are linear. Constraints (A3) ensure block b is not extracted at period t unless every block b' in b's direct predecessor set (B_b) is extracted at, or prior to, period t. Constraints (A4) enforce minimum resource requirements and maximum resource capacities in each period. Although our definition of r considers only two resources, this is for expository purposes only; without loss of generality, the formulation can accommodate other side constraints in the form of knapsacks. Regarding the data used in these knapsack constraints (i.e., a_{rb}, $\underline{e}_r$, $\bar{e}_r$), without loss of generality in the formulations we consider, we express the data in tonnages. Constraints (A5) prevent any block b from being extracted more than once, whereas constraints (A6) restrict all decision variables to be binary.

Appendix B. Improved Formulations

The ($CPIT$) decision variables y_{bt}, as defined in Equation (A1), assume a value of 1 if block b is extracted at time t, and 0 otherwise. We therefore designate the ($CPIT$) problem thus formulated and presented in objective function (A2) and constraints (A3) to (A6) as ($CPIT^{\text{at}}$). Although formulating ($CPIT$) with variables defined this way appears in previous work, for example, Dagdelen and Johnson (1986), Ramazan (2007), and Osanloo et al. (2008), the formulation may be tightened. Specifically, constraint (A3) prevents block b from being extracted in the single period t prior to b's direct predecessors b', which implies that block b may also not be extracted in all periods prior to t. Therefore, summing $y_{bt'}$ over all $t' \le t$, as in constraint (B1), is valid and is stronger than constraint (A3) because of the additional variables (with positive coefficients) included on the left side of the constraint

$$\sum_{t' \le t} y_{bt'} \le \sum_{t' \le t} y_{b't'} \quad \forall\, b \in B,\; b' \in B_b,\; t \in T. \tag{B1}$$

For an in-depth comparison of OPBS models and formulations, see Newman et al. (2010); regarding dominance of valid inequalities, see Wolsey (1998, p. 141). Replacing constraint (A3) in ($CPIT^{\text{at}}$) with constraint (B1) provides a tighter formulation, which we refer to as the *tightened at* formulation, and denote as ($CPIT^{\text{att}}$). This tightened formulation yields a lower (tighter) LP relaxation value for the root node, thereby leaving a smaller integrality gap for an integer programming solver to close.

Another formulation-tightening approach is to define decision variables that assume a value of 1 if block b is mined by period t and 0 otherwise, as in Expression (B2):

$$w_{bt} = \begin{cases} 1 & \text{if block } b \text{ is extracted by time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{B2}$$

This requires a variable substitution, where the *by* variable values may be translated to their corresponding *at* variable values using Equation (B3):

$$y_{bt} = w_{bt} - w_{b,\,t-1} \quad \forall\, t \in \{1, \ldots, T\}, \qquad \text{where } w_{b0} \equiv 0, \forall\, b \in B. \tag{B3}$$

For example, if a solution requires block b to be extracted in period $t = 3$ during a time horizon of $T = 5$, the *at* variable values for block b are:

$$y_{b1} = 0, \quad y_{b2} = 0, \quad y_{b3} = 1, \quad y_{b4} = 0, \quad y_{b5} = 0,$$

while the corresponding *by* variable values for block b are:

$$w_{b1} = 0, \quad w_{b2} = 0, \quad w_{b3} = 1, \quad w_{b4} = 1, \quad w_{b5} = 1.$$

The corresponding *by* formulation is as follows:

Objective Function:

$$(CPIT^{\text{by}}) \quad \max \sum_{b \in B} \sum_{t \in T} v_{bt} \left(w_{bt} - w_{b,t-1} \right) \tag{B4}$$

Constraints:

$$w_{bt} \le w_{b't} \quad \forall\, b \in B, b' \in B_b, t \in T, \tag{B5}$$

$$\underline{e}_r \le \sum_{b \in B} a_{rb} \left(w_{bt} - w_{b,t-1} \right) \le \bar{e}_r \quad \forall\, r \in R, t \in T, \tag{B6}$$

$$w_{b,t-1} \le w_{bt} \quad \forall\, b \in B, t \in T, \tag{B7}$$

$$w_{bt} \in \{0, 1\} \quad \forall\, b \in B, t \in T;\; w_{b0} \equiv 0 \;\forall\, b. \tag{B8}$$

The objective function value (B4) and constraints (B6) and (B8) of ($CPIT^{\text{by}}$) are identical to expressions (A2), (A4), and (A6) of ($CPIT^{\text{at}}$), respectively, where the *at* variables (y_{bt}) are replaced either with their *by* variable counterparts (w_{bt}) or with their transformation ($w_{bt} - w_{b,t-1}$). The number of precedence constraints (B5) equals that in constraint (A3); however, the *by* variable definition in constraints (B5) yields a less dense representation in the constraint matrix than that in constraint (A3). The problem ($CPIT^{\text{by}}$) does not require constraints equivalent to constraint (A5), but instead contains the additional constraints (B7), which enforce the *by* behavior. Although

($CPIT^{\text{by}}$) has an additional $|B| \cdot |T-2|$ constraints more than ($CPIT^{\text{at}}$) (where $|N|$ denotes the cardinality of the set N), our results show that ($CPIT^{\text{by}}$) provides a tighter LP relaxation bound than does ($CPIT^{\text{at}}$), and consequently improves overall performance when solving ($CPIT$).

The problems ($CPIT^{\text{by}}$) and ($CPIT^{\text{att}}$) produce the same LP relaxation objective function values; however, ($CPIT^{\text{by}}$) has a less dense constraint matrix than does ($CPIT^{\text{att}}$). More importantly, constraints (B5) make explicitly clear the underlying network structure. This insight, and use of the *by* variables, was first proposed in Johnson (1968).

Appendix C. Early Start Algorithm

Early Start (ES) Algorithm —Additional Notation

- S_{rb}: Total tonnage of blocks in the set $\mathscr{B}_b$ consuming operational resource r (tons).
- ES_{rb}: Earliest start time of block b based on the maximum capacity of operational resource r.
- ES_b: Earliest start time of block b based on the most constraining operational resource.

Early Start (ES) Algorithm

DESCRIPTION: An algorithm to calculate the earliest time a block may be extracted and, if applicable, processed, based on maximum resource capacities.

NOTE: We use the ceiling function ($\lceil a/b \rceil$), which rounds a/b up to the next largest integer if a/b is fractional, and returns a/b otherwise.

INPUT: B, $\bar{e}_r \geq 0$, and $a_{rb} \geq 0$, $\mathscr{B}_b$, $\forall b \in B$.

OUTPUT: $ES_b, \forall b \in B$, the earliest time block b may be extracted and, if applicable, processed, based on maximum resource capacities. The "$\leftarrow$" notation indicates the assignment of $\max_r\{ES_{rb}\}$ to ES_b.

```
{
    for (all blocks $b \in B$) {
        $S_{rb} = \sum_{b'' \in \mathscr{B}_b} a_{rb''}$
        $ES_{rb} = \lceil S_{rb}/\bar{e}_r \rceil$
        $ES_b \leftarrow \max(ES_{rb})$
    }
}
```

Note that we could relax the assumption of a constant operational resource capacity for all periods and, based on a generalization of the previous computation, define the early start time for block b as follows:

$$ES_b = \max_{r \in R} \min\left\{ t : \sum_{b'' \in \mathscr{B}_b} a_{rb''} \leq \sum_{t'=1}^{t} \bar{e}_{rt'} \right\}. \qquad \text{(C1)}$$

Appendix D. Enhanced Early Start Algorithm

Enhanced Early Start (EES) Algorithm—Additional Notation

- $T_{\underline{r}b}$: Number of periods for which the minimum resource requirement $\underline{e}_r$ is supportable by S_{rb} (periods).
- $\hat{e}_{1b}$: Residual production capacity after extracting S_{1b} in ES_b periods (tons).
- $\hat{e}_{2b}$: Residual processing requirement after extracting S_{2b} in ES_b periods (tons).
- EES_b: Period before which it is infeasible to extract block b given minimum resource requirement and maximum resource capacity constraints (period).

Enhanced Early Start (EES) Algorithm

DESCRIPTION: An algorithm to calculate the earliest time a block may be extracted and, if applicable, processed, considering the aggregate minimum and maximum resource capacities over ES_b periods.

NOTE: We use the *floor* function ($\lfloor a/b \rfloor$), which rounds a/b down to the next smallest integer if a/b is fractional, and returns a/b otherwise.

INPUT: B, $\bar{e}_1$, $\underline{e}_2$, and ES_b, S_{1b}, S_{2b}, $\forall b \in B$.

OUTPUT: EES_b, the time period before which it is infeasible to remove block b considering the aggregate minimum resource capacities, $\forall b \in B$.

```
{
    for (all blocks $b \in B$) {
        $T_{\underline{2}b} = \lfloor S_{2b}/\underline{e}_2 \rfloor$
        If $ES_b \leq T_{\underline{2}b}$ then {
            $EES_b \leftarrow ES_b$
        }
        Else {
            $\hat{e}_{1b} = \bar{e}_1 * ES_b - S_{1b}$
            $\hat{e}_{2b} = \underline{e}_2 * ES_b - S_{2b}$
            If $\hat{e}_{1b} < \hat{e}_{2b}$ then {
                $EES_b \leftarrow ES_b + 1$
            }
            Else {
                $EES_b \leftarrow ES_b$
            }
        }
    }
}
```

References

Achterberg T (2007) Constraint integer programming. Doctoral dissertation, Technical University Berlin, Berlin.

Amaya J, Espinoza D, Goycoolea M, Moreno E, Prevost T, Rubio E (2009) A scalable approach to optimal block scheduling. Accessed October 1, 2013, http://mgoycool.uai.cl/papers/09amaya_apcom.pdf.

AMPL (2009) AMPL: A modeling language for mathematical programming. Accessed October 1, 2013, www.ampl.com.

Bertsimas D, Weismantel R (2005) *Optimization Over Integers* (Dynamic Ideas, Belmont, MA).

Bixby R, Rothberg E (2007) Progress in computational mixed integer programming: A look back from the other side of the tipping point. *Ann. Oper. Res.* 149(1):37–41.

Boland N, Bley A, Fricke C, Froyland G, Sotirov R (2012) Clique-based facets for the precedence constrained knapsack problem. *Math. Programming* 133(1–2):481–511.

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

Cullenbine C, Wood R, Newman A (2011) A sliding time window heuristic for open pit mine block sequencing. *Optim. Lett.* 5(3):365–377.

Dagdelen K, Johnson T (1986) Optimum open pit mine production scheduling by Lagrangian parameterization. *Proc. 19th Internat. Appl. Comput. Oper. Res. Mineral Indust. (APCOM) Sympos.* (Society for Mining, Metallurgy and Exploration, Littleton, CO), 127–141.

De Kock P (2007) A back to basics approach to mining strategy formulation. Accessed October 1, 2013, http://www.saimm.co.za/Conferences/HMC2007/173-178_deKock.pdf.

Espinoza D, Goycoolea M, Moreno E, Newman A (2013) Minelib 2011: A library of open pit production scheduling problems. *Ann. Oper. Res.* 206(1):93–114.

Gaupp M (2008) Methods for improving the tractability of the block sequencing problem for open pit mining. Doctoral dissertation, Colorado School of Mines, Golden, CO.

Hanson N (2012) Whittle consulting provided via personal e-mail communication with A. Brickey, March 23.

IBM (2011) IBM ILOG CPLEX optimization studio v12.2. Accessed October 1, 2013, http://publib.boulder.ibm.com/infocenter/cosinfoc/v12r2/index.jsp/.

Johnson T (1968) Optimum open pit mine production scheduling. Doctoral dissertation, University of California, Berkeley.

Klotz E, Newman AM (2013a) Practical guidelines for solving difficult linear programs. *Surveys Oper. Res. Management Sci.* 18 (1–2): 1–17.

Klotz E, Newman AM (2013b) Practical guidelines for solving difficult mixed integer programs. *Surveys Oper. Res. Management Sci.* 18(1–2):18–32.

Lambert WB, Newman AM (2013) Tailored Lagrangian relaxation for the open pit block sequencing problem. *Ann. Oper. Res.*, ePub ahead of print January 16, http://link.springer.com/article/10.1007/s10479-012-1287-y/fulltext.html.

Lerchs H, Grossmann I (1965) Optimum design of open-pit mines. *Canadian Mining Metallurgical Bull.* 58(633):47–54.

Moreno E, Espinoza D, Goycoolea M (2010) Large-scale multi-period precedence constrained knapsack problem: A mining application. *Electronic Notes Discrete Math.* 36:407–414.

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

Osanloo M, Gholamnejad J, Karimi B (2008) Long-term open pit mine production planning: A review of models and algorithms. *Internat. J. Mining Reclamation Environ.* 22(1):3–35.

Park K, Park S (1997) Lifting cover inequalities for the precedence-constrained knapsack problem. *Discrete Appl. Math.* 72(3): 219–241.

Pochet Y, Wolsey L (2006) *Production Planning by Mixed Integer Programming* (Springer Verlag, New York).

Ramazan S (2007) The new fundamental tree algorithm for production scheduling of open pit mines. *Eur. J. Oper. Res.* 177(2): 1153–1166.

Somrit C (2011) Development of a new open pit mine phase design and production scheduling algorithm using mixed integer linear programming. Doctoral dissertation, Colorado School of Mines, Golden, CO.

Wolsey L (1998) *Integer Programming* (John Wiley & Sons, New York).

W. Brian Lambert is a senior consultant with Minemax, Inc. He holds a BA in general physics from the University of California, Los Angeles, an MS in operations research from the Naval Postgraduate School, and a PhD in mineral and energy economics from the Colorado School of Mines. Prior to joining Minemax, Inc., he consulted with life science companies as a decision and R&D portfolio analyst with Kromite, and was an operations analyst and naval aviator in the United States Marine Corps.

Andrea Brickey is a licensed professional mining engineer and is a PhD candidate in mining engineering at the Colorado School of Mines in Golden, Colorado. She consults for a global mining company and has worked extensively in Africa and North and South America mining copper, gold, silver, nickel, phosphate, and coal.

Alexandra M. Newman is an associate professor in the Division of Economics and Business at the Colorado School of Mines. She holds a BS in applied mathematics from the University of Chicago, an MS in operations research from the University of California, Berkeley, and a PhD in industrial engineering and operations research from the University of California, Berkeley. Prior to joining the Colorado School of Mines, she held the appointment of research assistant professor in the Operations Research Department at the Naval Postgraduate School. Her primary research interests lie in optimization modeling, particularly as it is applied to logistics, the energy and mining industries, and military operations. She is serving as associate editor of *Operations Research* and *Interfaces*, and is an active member of INFORMS.

Kelly Eurek is an energy analyst in the Strategic Energy Analysis Center at the National Renewable Energy Laboratory. He holds a BS in mechanical engineering and an MS in engineering and technology management from the Colorado School of Mines. His primary research interests lie in optimization modeling as it is applied to the electric power sector.