J Glob Optim (2015) 63:555–582
DOI 10.1007/s10898-014-0185-z

A hybrid method based on linear programming and variable neighborhood descent for scheduling production in open-pit mines

Amina Lamghari · Roussos Dimitrakopoulos · Jacques A. Ferland

Received: 30 January 2013 / Accepted: 31 March 2014 / Published online: 18 April 2014

Abstract Production scheduling of open-pit mines is an important problem arising in surface mine planning as it determines the raw materials to be produced yearly over the life of the mine, assesses the value of the mine, and contributes to the sustainable utilization of mineral resources. Finding the optimal schedule is a complex task, involving large data sets and multiple constraints. This paper introduces a two-phase hybrid solution method. The first phase relies on solving a series of linear programming problems to generate an initial solution. In the second phase, a variable neighborhood descent procedure is applied to improve the solution. Upper bounds provided by CPLEX are used to evaluate the efficiency of the proposed method. Its performance is also assessed by comparing it to recent solution methods proposed in the literature and to an alternate method implemented in commercial mine planning software commonly used by professional mine planners. The results of these computational experiments indicate the efficiency of the proposed method and its superiority over the other methods. It finds excellent solutions (within less than 3.2 % of optimality on average) for large instances of the problem in a few seconds up to a few minutes. It also provides new best-known solutions for benchmark instances from the literature, and it can solve instances recently-published algorithms have found intractable.

Keywords Variable neighborhood descent · Hybrid methods · Scheduling · Open-pit mine planning

A. Lamghari (✉) · R. Dimitrakopoulos
COSMO - Stochastic Mine Planning Laboratory, Department of Mining and Materials Engineering, McGill University, FDA Building, 3450 University Street, Montreal, Quebec H3A 2A7, Canada
e-mail: amina.lamghari@mcgill.ca

R. Dimitrakopoulos
e-mail: roussos.dimitrakopoulos@mcgill.ca

J. A. Ferland
Department of Computer Science and Operations Research, University of Montreal, C.P. 6128, SUCC. Centre-Ville, Montreal, Quebec H3C 3J7, Canada
e-mail: ferland@iro.umontreal.ca

1 Introduction

Production scheduling of open-pit mines is an important issue in surface mine planning as it determines the raw materials to be produced yearly over the life of the mine, assesses the value of the mine, and contributes to the sustainable utilization of mineral resources. The problem is complex as it involves large data sets and multiple constraints, placing a strain on computational resources.

The mineral deposit is represented as a three-dimensional array of blocks. Each block has a weight and a metal content estimated using information obtained from drilling. To recover the metal, the block must first be extracted from the ground and then treated in a plant. These operations are termed mining and processing, respectively. The set of blocks can be divided into two distinct subsets: the set of *ore* blocks sent to the plant (i.e., blocks that are processed to produce metal), and the set of *waste* blocks formed by the remaining blocks. *Waste* blocks are not processed, but the physical nature of the problem requires mining them in order to have access to *ore* blocks. Blocks that should be removed to have access to a given block are called its *predecessors*. Each block also has an economic value representing the net profit associated with it. Hence, for an *ore* block, this value is equal to the selling revenue less the mining and the processing costs. For a *waste* block, it is equal to minus the cost of mining the block.

The open-pit mine production scheduling problem (MPSP), also known as the open-pit mine block sequencing problem or the constrained pit limit problem, consists of identifying which blocks should be mined during each period of the life of the mine so as to maximize the net present value (the total discounted profit) of the mining operation. Decisions on block scheduling are subject to various constraints, typically:

- *Reserve constraints*: a block can be mined at most once during the horizon.
- *Slope constraints*: a block cannot be mined before its predecessors.
- *Mining constraints*: the total weight of blocks (*waste* and *ore*) mined during each period of the horizon must not exceed the available extraction equipment capacity, referred to as the mining capacity.
- *Processing constraints*: the total weight of *ore* blocks processed during each period of the horizon must not exceed the processing plant capacity.

The MPSP can be formulated as a combinatorial optimization problem. Because of its complexity and its practical interest, it has been widely studied since the 1960s [21]. For an overview of the different formulations and solution methodologies, see [23]. As this review paper shows, different solution approaches have been proposed in the literature. These include exact methods [9,13], heuristics [10,17], and metaheuristics [14,16]. Each approach has strengths and weaknesses.

Exact methods solve optimally the MPSP, but their major limitation is that they can only be applied to instances of relatively small size. Solving instances of realistic size, where the number of blocks is typically in the order of tens to hundreds of thousands, requires prohibitive computational times. To reduce the number of binary variables and thus make larger instances computationally tractable by exact methods, Ramazan and Dimitrakopoulos [25] propose a mixed integer programming formulation where only the variables associated with *ore* blocks are restricted to be binary. Another approach for handling the large number of binary variables exploits the structure of the problem to aggregate blocks into groups, leading to a reduction in the number of variables [5,24]. Aggregation, however, can severely compromise the validity and usefulness of the solution [3]. It causes loss of profitability and may even lead to infeasible solutions [5]. In order to reduce the number of binary variables, Bley et al. [4] address a

different approach involving cutting planes. The authors add to the integer programming formulation inequalities derived by combining the constraints of the problem to eliminate a number of decision variables from the model prior to optimization. Their results indicate that adding such inequalities is, in the majority of the cases tested, beneficial for reducing the CPU time required by the solver. However, their experiments were conducted on instances containing only hundreds of blocks and 5 or 10 periods.

Heuristics and metaheuristics can tackle large instances of the MPSP in a reasonable amount of time, but they do not guarantee optimality. Their efficiency and robustness can be improved by combining them with other techniques. A number of such hybrid methods have been developed over the years. Sevim and Lei [26] and Tolwinski and Underwood [27] combine heuristics with dynamic programming techniques. More recent approaches combine heuristics with exact methods. Moreno et al. [22] introduce an algorithm for solving the linear relaxation of the MPSP and an LP-based heuristic to obtain feasible solutions. However, the algorithm proposed by the authors to solve the linear relaxation is only applicable to the variant of the MPSP with a single resource constraint per period and for which such a constraint is an upper bound (i.e., referring to the description given in the beginning of this section, the variant of the MPSP including either the *mining* or the *processing constraints* in addition to the *slope constraints*). Bienstock and Zuckerberg [3] propose another algorithm for solving the linear relaxation of the MPSP that can handle any number of resource constraints.

Another hybrid approach for the MPSP is the one by Amaya et al. [1]. Starting from an initial feasible solution generated using Gershon's heuristic [17], the authors iteratively fix parts of the incumbent solution and re-optimize the "unfixed" parts. This defines an integer programming sub-problem at each iteration that is solved exactly, using the commercial solver CPLEX. Chicoisne et al. [11] use a similar approach. They first use the method in [22] to generate a feasible solution, followed by an improvement integer programming-based heuristic, which is an enhanced version of that in [1]. Cullenbine et al. [12] also solve a series of mixed-integer programs that have fixed variables. The authors consider, however, a variant of the MPSP incorporating lower bounds on mining and processing, which is, as noted by the authors, harder to solve than the variant where the lower bounds are omitted (i.e., the variant described in the beginning of this section and considered by Chicoisne et al. [11]). The drawback of the recent hybrid algorithms [11,12] is that they rely on time consuming integer programming algorithms. The method in [11] can solve instances with up to five million blocks and 15 years but might require 8 h to improve the solution and cannot handle lower bound constraints. On the other hand, the method in [12] can handle lower bound constraints but has been able to tackle only instances with up to 25,000 blocks and 15 periods.

In this paper, we propose a variable neighborhood descent based method for the MPSP. Variable neighborhood descent (VND) is a variant of variable neighborhood search [18]. The basic idea is the same: neighborhood change to escape from local optima, but the different neighborhoods are explored using a best improvement local descent. VND has been shown effective in solving a variety of combinatorial optimization problems (see [8] for an example). To generate the initial solution to be improved by the VND procedure, we develop a decomposition based heuristic. The basic idea is to reduce the complexity of the problem by exploiting its structure and decomposing it into easier to solve sub-problems. Each sub-problem is associated with one period, and solving it consists of solving a linear program, not an integer or a mixed-integer program. This is followed by a repair heuristic to make the solution of the sub-problem feasible if necessary. When all sub-problems are considered, their solutions are combined to obtain a feasible solution of the original problem (the initial solution).

We applied the method described above to solve two different variants of the MPSP: the variant described in the beginning of this section, commonly studied in the literature, and the variant incorporating lower bounds on mining and processing, considered in [12] and known to be more difficult to solve. For both variants, the computational results indicate that the proposed solution method is very effective and robust, providing, for all the tested instances, near-optimal solutions in a few seconds up to a few minutes. It generally outperforms recently-proposed solution methods for the MPSP and provides new best-known solutions for benchmark instances from the literature.

From a broad perspective, the method proposed in this paper is a hybrid method combining mathematical programming algorithms with heuristic search techniques, able to tackle large instances of the MPSP. In this sense, it is similar to recent approaches in the literature. However, it has key characteristics which differentiate it from existing methods. First, it doesn't resort to aggregation to tackle large instances of the problem. Second, it doesn't rely on solving a series of integer or mixed integer sub-problems but rather exploits the problem structure and uses variable neighborhood descent to quickly find improving solutions. When generating the initial solution, it uses an exact method to solve linear programs. Because of that, its running time has a slower growth rate compared to recent solution methods in the literature. Third, it is not a specialized method tailored to solve one variant of the MPSP, but it can be easily adapted to account for additional constraints. Last but not least, it doesn't require any external software, such as CPLEX, in order to be implemented. Indeed, although CPLEX is used in this study to solve the linear programs when generating the initial solution, any maximum flow algorithm, such as the pseudo-flow algorithm proposed by Hochbaum [19], can be used.

Note that this paper deals with the deterministic version of the MPSP, which assumes that all the problem parameters are well known. There exist other versions where one or some components of the problem (e.g., prices, metal content, etc.) are not known with certainty. These stochastic versions of the MPSP have received increasing attention in recent years (see for instance, [2,6,7]).

The remainder of the paper is organized as follows: In Sect. 2 a mathematical formulation of the problem is provided. In Sect. 3 the heuristic for generating an initial solution is outlined. In Sect. 4 the components of the variable neighborhood descent heuristic used to improve the solution are described. The results of an extensive computational study are presented in Sect. 5. This is followed by conclusions in Sect. 6.

2 Mathematical formulation

As mentioned in the introduction, different mathematical formulations of the MPSP have been proposed in the literature. In this paper, the integer linear programming formulation proposed by Caccetta and Hill [9] is used. The following notation is used to specify the mathematical model:

- T: the scheduling horizon.
- t: period index, $t = 1, \ldots, T$.
- N: the number of blocks.
- i: block index, $i = 1, \ldots, N$.
- $\mathcal{P}_i$: the set of predecessors of block i; i.e., blocks that have to be removed to have access to block i.
- $\mathcal{S}_i$: the set of successors of block i; i.e., $s \in \mathcal{S}_i$ if $i \in \mathcal{P}_s$.

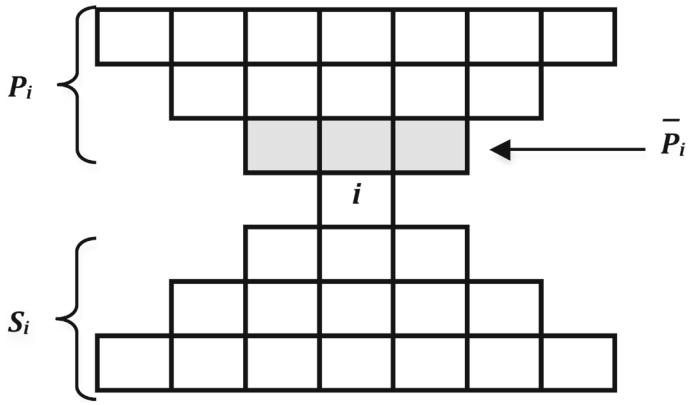

Fig. 1 Illustration of the sets $\mathcal{P}_i$, $\mathcal{S}_i$, and $\overline{\mathcal{P}}_i$ when precedences are defined by a 45-degree slope angle and one level above

- $\overline{\mathcal{P}}_i$: the set of immediate predecessors of block i. The relationship between block precedences is transitive; i.e., if block j is a predecessor of block i, and if block k is a predecessor of block j, then block k is also a predecessor of block i. This transitivity property is used to describe the immediate predecessors as being those that are not implied by any other pair of precedences. In the example above, k is not an immediate predecessor of i.
  Figure 1 gives a 2-dimensional illustration of the sets $\mathcal{P}_i$, $\mathcal{S}_i$, and $\overline{\mathcal{P}}_i$ when precedences are defined by a 45-degree slope angle and one level above.
- w_i: the weight of block i (typically in tons).
- θ_i: a parameter indicating the group of block i
  $\theta_i = \begin{cases} 1 \text{ if block } i \text{ is an } ore \text{ block,} \\ 0 \text{ otherwise (i.e., if } i \text{ is a } waste \text{ block).} \end{cases}$
- v_i: the economic value of block i.
- d: the discount rate per period.
- $v_i^t = \dfrac{v_i}{(1+d)^t}$: the discounted economic value of block i if mined in period t. Note that we assume that *ore* blocks are processed during the same period that they are mined and that the profit is also generated during that period.
- M^t: the mining capacity available at period t.
- Θ^t: the processing capacity available at period t.
- A binary variable is associated with each block i for each period t:
  $x_i^t = \begin{cases} 1 \text{ if block } i \text{ is mined by period } t, \\ 0 \text{ otherwise.} \end{cases}$
  This means that if block i is mined in period τ, then $x_i^t = 0$ for all $t = 1, \ldots, \tau - 1$ and $x_i^t = 1$ for all $t = \tau, \ldots, T$. If i is not mined during the horizon, then $x_i^t = 0$ for all $t = 1, \ldots, T$.

The mathematical model can be summarized as follows:

$$\max \sum_{i=1}^{N} v_i^1 x_i^1 + \sum_{t=2}^{T} \sum_{i=1}^{N} v_i^t \left(x_i^t - x_i^{t-1} \right) \tag{1}$$

$$(MM) \qquad \textbf{Subject to}$$

$$x_i^{t-1} \leq x_i^t \qquad i = 1, \ldots, N, \;\; t = 2, \ldots, T \tag{2}$$

$$x_i^t \leq x_p^t \qquad i = 1, \ldots, N, \quad p \in \overline{\mathcal{P}}_i, \quad t = 1, \ldots, T \tag{3}$$

$$\sum_{i=1}^{N} w_i x_i^1 \leq M^1 \tag{4}$$

$$\sum_{i=1}^{N} w_i \left(x_i^t - x_i^{t-1}\right) \leq M^t \qquad t = 2, \ldots, T \tag{5}$$

$$\sum_{i=1}^{N} \theta_i w_i x_i^1 \leq \Theta^1 \tag{6}$$

$$\sum_{i=1}^{N} \theta_i w_i \left(x_i^t - x_i^{t-1}\right) \leq \Theta^t \qquad t = 2, \ldots, T \tag{7}$$

$$x_i^t = 0 \text{ or } 1 \qquad i = 1, \ldots, N, \quad t = 1, \ldots, T. \tag{8}$$

The objective function (1) to be maximized is equal to the net present value of the mining operation. Constraints (2) guarantee that each block i is mined at most once during the horizon (*reserve constraints*). The mining precedence (*slope constraints*) is enforced by constraints (3). Constraints (4) and (5) impose an upper bound M^t on the amount of material (*waste* and *ore*) mined during each period t (*mining constraints*). Constraints (6) and (7) ensure that the amount of *ore* processed during each period does not exceed the processing capacity available at that period (*processing constraints*).

The solution procedure, which includes two phases, is described in the next sections.

3 Phase 1 to construct an initial feasible solution (SH)

In this section, a general overview of the algorithm used to generate an initial feasible solution is first presented. This is followed by a step-by-step description of it.

3.1 Overview of the algorithm

The sub-problems associated with the periods t ($t = 1, \ldots, T$) are solved sequentially (in increasing order of t), and the solutions of the sub-problems are combined to generate the initial solution. The procedure to deal with the sub-problem associated with each period t can be summarized as follows: First, a set of blocks to be mined in t is identified by solving a linear programming model. This solution satisfies the *slope constraints*, but it may violate the *mining constraints* and/or the *processing constraints*. In this case, a repair heuristic is applied to modify the solution in order to satisfy these constraints.

3.2 Step 1 solving the sub-problem for period t

We specify and solve a linear programming sub-problem to determine a set of blocks $\mathcal{B}^t$ to be mined in period t.

The *reserve* and the *slope constraints* are considered to specify the constraints of the sub-problem. Let us denote by $\mathcal{R}^t$ the set of blocks not mined at the beginning of period t ($\mathcal{R}^1 = \{1, \ldots, N\}$ and $\mathcal{R}^t = \mathcal{R}^{t-1} \setminus \mathcal{B}^{t-1}$ if $t = 2, \ldots, T$). In order to satisfy the *reserve constraints*, the blocks to be included in $\mathcal{B}^t$ should be selected from $\mathcal{R}^t$. Consider

any candidate block $i \in \mathcal{R}^t$. The *slope constraints* require that to include i in $\mathcal{B}^t$, we must also include all blocks $j \in \mathcal{N}_i = \mathcal{P}_i \cap \mathcal{R}^t$ (the set of blocks that are predecessors of i and not mined yet).

To specify the objective function of the sub-problem, we consider the *mining* and the *processing constraints*. Indeed, i should not be included in $\mathcal{B}^t$ if $w_i + \sum_{j\in\mathcal{N}_i} w_j > M^t$ or $\theta_i w_i + \sum_{j\in\mathcal{N}_i} \theta_j w_j > \Theta^t$ because this would lead to violation of the *mining constraints* or the *processing constraints*. The economic value v_i of such a block i is thus modified and set to a large negative value to penalize its extraction. Note that we extend this penalty of extraction to any block $i \in \mathcal{R}^t$ such that $w_i + \sum_{j\in\mathcal{N}_i} w_j > \alpha M^t$ or $\theta_i w_i + \sum_{j\in\mathcal{N}_i} \theta_j w_j > \alpha\Theta^t$, where α is a random number in the interval $[\alpha_1, \alpha_2]$ if $t < T$, and $\alpha = 1$ if $t = T$. Moreover, α_1 and α_2 are parameters of the procedure in the interval $]0, 1]$.

For the other blocks $i \in \mathcal{R}^t$, v_i is also modified but in order to favor blocks having high value per unit of weight. This can be summarized as follows ($\bar{v_i}$ denotes the modified economic value of block i):

$$\bar{v_i} = \begin{cases} \dfrac{v_i}{w_i} & \text{if } w_i + \displaystyle\sum_{j\in\mathcal{N}_i} w_j \le \alpha M^t \text{ and } \theta_i w_i + \sum_{j\in\mathcal{N}_i} \theta_j w_j \le \alpha\Theta^t, \\ -C & \text{otherwise.} \end{cases}$$

The sub-problem associated with period t can then be summarized as follows:

$$\max \sum_{i\in\mathcal{R}^t} \bar{v_i} y_i \tag{9}$$

$$(SP^t) \qquad \textbf{Subject to}$$

$$y_i - y_j \le 0 \qquad i \in \mathcal{R}^t,\ j \in \overline{\mathcal{P}}_i \cap \mathcal{R}^t \tag{10}$$

$$0 \le y_i \le 1 \qquad i \in \mathcal{R}^t. \tag{11}$$

Consider an optimal solution y^* for the linear programming model (SP^t). Since the constraints matrix of (SP^t) is unimodular, y^* is integer (i.e., $y_i^* = 0$ or $1 \quad \forall i \in \mathcal{R}^t$). For each block $i \in \mathcal{R}^t$, if $y_i^* = 1$, we include i in $\mathcal{B}^t$ (i.e., $x_i^\tau := 1\ \forall \tau = t, \ldots, T$). All the other blocks (i.e., blocks i such that $y_i^* = 0$) are inserted in the set $\mathcal{R}^{t+1}$.

Note that the algorithm described above can be seen as using logical implications of the constraints to exclude some blocks, and then, considering the remaining blocks, to find an ultimate pit, but to do that, instead of considering the economic value of each block, the value per unit of weight is considered. In our tests, we found that using the normalized value (i.e., value per unit of weight) leads to better results compared to using the original value.

Even if the *mining* and *processing constraints* are used partly to discard some blocks from $\mathcal{B}^t$ (those whose modified economic values are set to $-C$), $\sum_{i\in\mathcal{B}^t} w_i$ (respectively, $\sum_{i\in\mathcal{B}^t} \theta_i w_i$) may exceed the mining (respectively, the processing) capacity available at period t. We therefore introduce a heuristic allowing us to satisfy the *mining* and the *processing constraints* for period t (if they are violated).

3.3 Step 2 to satisfy the mining and the processing constraints

We use a sequential heuristic procedure where at each iteration a block is removed from the set $\mathcal{B}^t$ and added to the set $\mathcal{R}^{t+1}$. Let us analyze a typical iteration.

Consider the set $\mathcal{E} = \{i \in \mathcal{B}^t : s \notin \mathcal{B}^t\ \forall s \in \mathcal{S}_i\}$ of blocks $i \in \mathcal{B}^t$ having no successors in $\mathcal{B}^t$. Clearly, only these blocks can be removed from $\mathcal{B}^t$ while satisfying the *slope constraints*. For each candidate block $i \in \mathcal{E}$, let $f_i = \dfrac{v_i}{w_i} + \displaystyle\sum_{p\in\mathcal{P}_i\cap\mathcal{B}^t} \frac{v_p}{w_p}$ be the total unit economic

value of i and its predecessors that belong to $\mathcal{B}^t$. Select the block $i^* \in \mathcal{E}$ minimizing the value of f_i. Ties are broken up randomly. Remove i^* from $\mathcal{B}^t$, add it to $\mathcal{R}^{t+1}$, and update $\mathcal{E}$. Note that the rule used to select the blocks in $\mathcal{E}$ induces some look-ahead features to remove less valuable blocks from $\mathcal{B}^t$. However, it does not guarantee that a feasible solution of good quality will be obtained.

This process is repeated until the *mining* and the *processing constraints* are approximately satisfied; i.e., until

$$\sum_{i \in \mathcal{B}^t} w_i \leq \beta M^t \tag{12}$$

$$\sum_{i \in \mathcal{B}^t} \theta_i w_i \leq \beta \Theta^t \tag{13}$$

where β is a random number in the interval $[\beta_1, \beta_2]$, and β_1 and β_2 are parameters of the procedure in]0, 1].

4 Phase 2 to improve the solution (VND)

As mentioned before, the solution $x = \cup_{t=1}^{T} \mathcal{B}^t$ generated in Phase 1 is feasible, and it is improved by applying an adaptation of the Variable Neighborhood Descent method (VND) proposed by Hansen and Mladenovic [18]. The basic idea of VND is to combine different descent heuristics based on different neighborhood structures to escape from local optima. In the following, we first describe the neighborhood structures used in our adaptation of the VND method. Next, we outline the procedure used to improve the solution x.

4.1 Neighborhood structures

The following three neighborhood structures are used in our adaptation of the VND method:

- N^1 (*Exchange*): Let i and j be two blocks mined in periods t and $(t+1)$, respectively. An exchange consists of replacing $\mathcal{B}^t$ and $\mathcal{B}^{t+1}$ by $(\mathcal{B}^t - \{i\}) + \{j\}$ and $(\mathcal{B}^{t+1} - \{j\}) + \{i\}$, respectively. The exchange of two blocks is feasible if the resulting solution is feasible; i.e., only if it satisfies the *slope*, the *mining*, and the *processing constraints*. Figure 2 gives a 2-dimensional illustration of an *exchange* move involving two blocks, i and j, with $T = 2$.
- N^2 (*Shift-after*): Let i be a block mined in period t, and let $\mathcal{I} = \{i\} \cup \{\text{block } s : s \in \mathcal{S}_i \cap \mathcal{B}^t\}$ denote the set including i and its successors mined in the same period. A shift-after consists

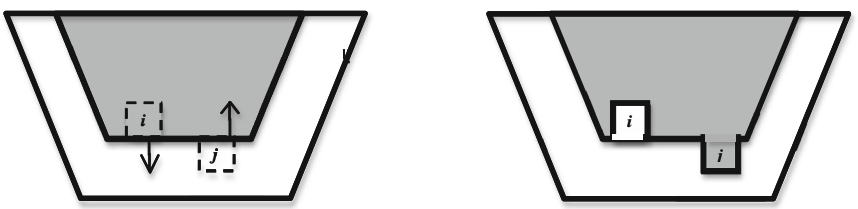

Fig. 2 *Exchange* move between blocks i and j with $T = 2$. The *grey area* represents the set of blocks to be mined in the first period ($\mathcal{B}^1$), while the *white area* delimited by the *thick lines* represents the set of blocks to be mined in period 2 ($\mathcal{B}^2$)

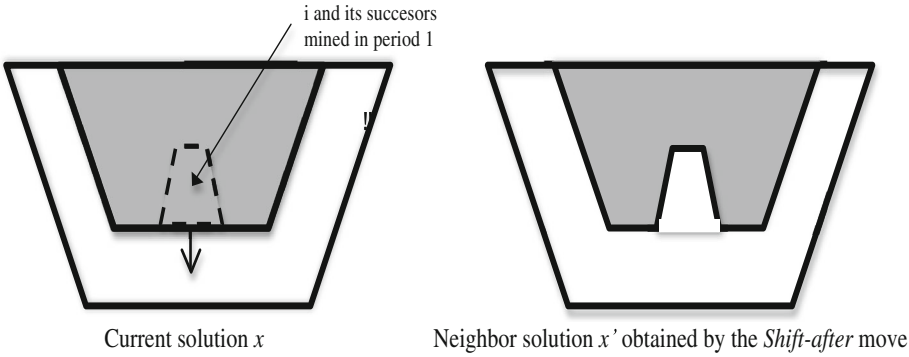

Fig. 3 *Shift-after* move of block i and its successors mined in the same period. The *grey area* represents the set of blocks to be mined in the first period ($\mathcal{B}^1$), while the *white area* delimited by the *thick lines* represents the set of blocks to be mined in period 2 ($\mathcal{B}^2$)

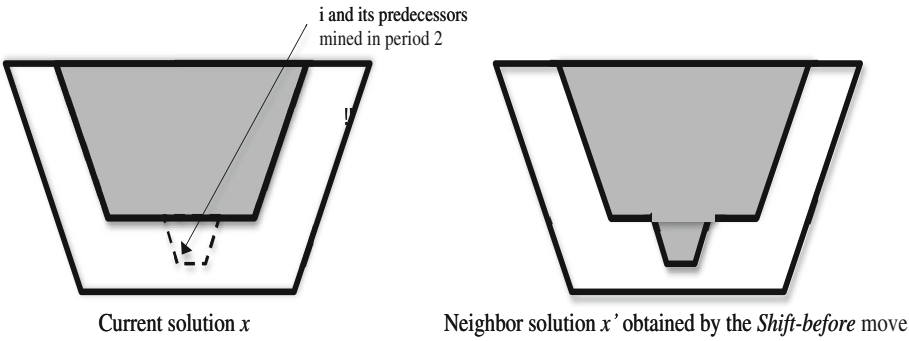

Fig. 4 *Shift-before* move of block i and its predecessors mined in the same period. The *grey area* represents the set of blocks to be mined in the first period ($\mathcal{B}^1$), while the *white area* delimited by the *thick lines* represents the set of blocks to be mined in period 2 ($\mathcal{B}^2$)

of replacing $\mathcal{B}^t$ and $\mathcal{B}^{t+1}$ by $\mathcal{B}^t - \mathcal{I}$ and $\mathcal{B}^{t+1} + \mathcal{I}$, respectively. Clearly, the *slope constraints* are satisfied in the resulting solution since the blocks are moved along with their successors. However, the *mining* and the *processing constraints* must be satisfied in period $(t+1)$ in order to allow this shift-after. Figure 3 illustrates the *Shift-after* move where block i and its successors mined in period 1 are moved to period 2.

– N^3 (*Shift-before*): Let i be a block mined in period t, and let $\mathcal{I} = \{i\} \cup \{\text{block } p \ : \ p \in \mathcal{P}_i \cap \mathcal{B}^t\}$ denote the set including i and its predecessors mined in the same period. A shift-before consists of replacing $\mathcal{B}^t$ and $\mathcal{B}^{t-1}$ by $\mathcal{B}^t - \mathcal{I}$ and $\mathcal{B}^{t-1} + \mathcal{I}$, respectively. As for the *Shift-after* neighborhood, the *slope constraints* are necessarily satisfied, but the *mining* and the *processing constraints* in period $(t-1)$ must be satisfied in order to allow this shift-before. Figure 4 illustrates the *Shift-before* move where block i and its predecessors mined in period 2 are moved to period 1.

The strategy to explore any of the three neighborhoods is as follows: Consider the first neighborhood N^1. Periods $t = 1, \ldots, (T-1)$ are considered sequentially in increasing order. Given a period t, all feasible exchanges involving pairs of blocks mined in t and $(t+1)$ are systematically considered. The best exchange is selected. We apply the selected exchange if it leads to a better solution or to a solution of the same value as the current solution (in order

to escape from the current local optimum). The process is iterated with the new solution. When no feasible exchange exists to further improve the solution or to get a solution of equal value, the next period $(t+1)$ is considered; i.e., exchanges involving pairs of blocks mined in periods $(t+1)$ and $(t+2)$ are evaluated.

The same exploration strategy is used when considering the *Shift-after* neighborhood N^2 except that periods $t = 1, \ldots, (T-1)$ are considered in decreasing order. For the *Shift-before* neighborhood N^3, periods $t = 2, \ldots, T$ are considered in increasing order.

4.2 Variable neighborhood descent procedure

The rules of a basic VND are applied: Start by exploring the *Exchange* neighborhood (N^1). When the search of N^1 is completed (i.e., for all $t = 1, \ldots, (T-1)$, no feasible exchange between pairs of blocks mined in periods t and $(t+1)$ exists to further improve the solution or to get a solution of equal value), restart a new search using the *Shift-after* neighborhood (N^2). Once the search of N^2 is completed, if the solution has been improved, return to N^1; otherwise, use the *Shift-before* neighborhood (N^3). This process terminates when no move in any of the three neighborhoods improves the value of the objective function.

Note that the *Shift-after* neighborhood (N^2) is explored before the *Shift-before* neighborhood (N^3) to create an opportunity for other blocks to be added to a given period t without exceeding the mining and the processing capacities available at t. Indeed, by first moving blocks from period 1 to period 2 (last shifts using N^2), more capacity becomes available in period 1 to include other blocks from period 2 (first shifts using N^3).

5 Numerical results

The method described in this paper is tested on instances generated from actual mineral deposits, as well as on benchmark instances from the literature. All numerical experiments were performed on an Intel(R) Xeon(R) CPU E7-8837 computer (2.67 GHz) with 1 TB of RAM running under Linux. Before reporting the numerical results, we first introduce the instances and the parameters used in the experiments.

5.1 Test instances

5.1.1 Instances generated from actual mineral deposits

Two different sets of instances, P_1 and P_2, are used. Instances in P_1 and P_2 are generated from an actual copper deposit where blocks i are of size $20 \times 20 \times 10$ meters and weigh $w_i = 10,800$ tons each, and from an actual gold deposit where blocks i are of size $15 \times 15 \times 10$ meters and weigh $w_i = 5,625$ tons each, respectively. The economic parameters used to compute the blocks' economic values v_i are also based on real-life data, and they are summarized in Table 1.

Each set (P_1 and P_2) includes 5 different instances characterized by a number of blocks N and a number of periods T, specified in Table 2. The 10 instances have been constructed by varying (perturbing) the blocks' economic values v_i. More precisely, to generate a specific instance, the profit associated with each block i in the mineral deposit ($i = 1, \ldots, \overline{N}$) is obtained by reducing v_i by a factor D. Then, the following mathematical model is solved to determine the pit limits; i.e., to identify the set of blocks maximizing the total profit but accounting only for the *slope constraints*:

Table 1 Economic parameters

Parameters	P_1	P_2
Mining cost	\$1/t	\$1/t
Processing cost	\$9/t	\$15/t
Metal price	\$2/lb	\$900/oz
Selling cost	\$0.3/lb	\$7/oz
Discount rate	10 %	10 %

Table 2 Characteristics of the 10 instances generated from actual mineral deposits

Set	Instance	D	Number of blocks (N)	Number of periods (T)
P_1	C1	20,000	4,273	3
Metal type: copper	C2	15,000	7,141	4
Block size: 20 × 20 × 10 m	C3	10,000	12,627	7
Block weight: w_i = 10,800 tons	C4	5,000	20,626	10
	C5	0	26, 021	13
P_2	G1	20,000	18,821	5
Metal type: gold	G2	15,000	23,901	7
Block size: 15 × 15 × 10 m	G3	10,000	30,013	8
Block weight: w_i = 5,625 tons	G4	5,000	34,981	9
	G5	0	40,762	11

$$\max \sum_{i=1}^{\overline{N}} (v_i - D) y_i \tag{14}$$

(PL_D) **Subject to**

$$y_i \leq y_p \qquad i = 1, \ldots, \overline{N},\ p \in \overline{\mathcal{P}}_i \tag{15}$$

$$y_i = 0 \text{ or } 1 \qquad i = 1, \ldots, \overline{N}. \tag{16}$$

Referring to an optimal solution y^* of (PL_D), $N = |\{\text{block } i \ : \ y_i^* = 1\}|$. Note that N decreases with the factor D. Each period is one year long. The number of periods is set to $T = \lceil \frac{\sum_{i=1}^{N} w_i}{22.3M} \rceil$ (M = million).

For each of the 10 instances, the production capacities are identical in all periods and emulate those in real-world problems. For each period t, the mining capacity W^t is set to $\lceil 1.25 \frac{\sum_{i=1}^{N} w_i}{T} \rceil$ and $\lceil 1.30 \frac{\sum_{i=1}^{N} w_i}{T} \rceil$ for instances in P_1 and P_2, respectively (i.e., $\frac{\text{total amount of rock}}{\text{number of periods}}$ plus a margin of 25 % and 30 %, respectively). The processing capacity Θ^t is set to $\lceil 1.05 \frac{\sum_{i=1}^{N} w_i \theta_i}{T} \rceil$ $\left(\text{i.e, } \frac{\text{total amount of } ore}{\text{number of periods plus a margin of}} 5\,\%\right)$.

5.1.2 First set of benchmark instances

As mentioned in the introduction, recent methods for solving the open-pit mine production scheduling problem (MPSP) have been proposed by Amaya et al. [1], Moreno et al. [22], and Chicoisne et al. [11]. These methods have been applied to a set of four instances. In our experiments, we use two of these four instances; namely, AmericaMine and Marvin. Note that Marvin is also used in the studies by Bienstock and Zuckerberg [3] and Cullenbine et

Table 3 Characteristics of the two instances in the first set of benchmark instances

Instance	Number of blocks (N)	Number of periods (T)
AmericaMine	19,320	15
Metal type: polymetalic (not specified by the authors)		
Block size: Unknown		
Block weight w_i: variable		
Marvin	53,668	15
Metal type: copper and gold		
Block size: $30 \times 30 \times 30$ m		
Block weight w_i: variable		

al. [12]. We don't use the other two instances (AsiaMine and Andina) because they were not available in the public domain (due to confidentiality considerations).

The characteristics of the two instances considered in this paper are as follows and are summarized in Table 3: AmericaMine is a hard rock polymetalic mine containing 19,320 blocks, while Marvin is a fictitious copper and gold orebody, included in Whittle [28], and has 53,668 blocks (Whittle is a commercial mine planning software commonly used by professional mine planners). In both instances, blocks have different weights and are scheduled over 15 years. The production capacities are identical in all years. For AmericaMine, the mining capacity W^t is equal to $1M$ and the processing capacity Θ^t is equal to $0.55M$ (M = million). For Marvin, $W^t = 60M$ and $\Theta^t = 20M$.

5.1.3 Second set of benchmark instances: MineLib test instances

The last dataset on which the method proposed in this paper is tested consists of instances from *MineLib*, a library of publicly available test problem instances proposed by Espinoza et al. [15] for three classical types of open pit mining problems. The authors refer to the variant of the open-pit mine production scheduling problem, denoted in this paper by MPSP, as the constrained pit limit problem, and they denote it by CPIT. They provide 11 instances where the number of operational resource constraints ranges from 1 to 3, and they are related to the total amount extracted and/or the total amount processed in one or two different mills. In 3 of these 11 instances (namely, D, McLaughlin Limit, and McLaughlin) data related to the weight of the blocks (w_i) is only available for *ore* blocks (i.e., data is missing for *waste* blocks). Therefore, the method described in Sect. 3 cannot be applied to generate the initial solution for these instances (recall that this method is based on the value per unit of weight of each block; i.e., $\frac{v_i}{w_i}$). Hence, these instances are not used in our experiments, and we consider only the 8 other instances in *MineLib*. Table 4 reports the size of each of these 8 instances; their other characteristics can be found at http://mansci.uai.cl/minelib. Note that the instance "marvin in Table 4 is different from the instance "marvin" in Table 3 as far as the number of blocks (N) and the number of periods (T) are concerned.

5.2 Parameter calibration

Version 12.5 of the commercial solver CPLEX was used to solve the sub-problems (SP^t); i.e., the mathematical model (9)–(11) introduced in Sect. 3.2. The *predual parameter* of CPLEX was set to 1; that is, the dual linear programming problem is passed to the optimizer.

Table 4 Characteristics of the eight instances in the second set of benchmark instances (instances from *MineLib*)

Instance	Number of blocks (N)	Number of periods (T)
newman1	1,060	6
Metal type: Unknown		
Block size: Unknown		
Block weight w_i: variable		
zuck-small	9,400	20
Metal type: Unknown		
Block size: Unknown		
Block weight w_i: variable		
zuck-medium	29,277	15
Metal type: Unknown		
Block size: Unknown		
Block weight w_i: variable		
p4hd	40,947	10
Metal type: gold and copper		
Block size: $50 \times 50 \times 20$ ft		
Block weight w_i: variable		
marvin	53,271	20
Metal type: gold and copper		
Block size: $30 \times 30 \times 30$ m		
Block weight w_i: variable		
w23	74,260	12
Metal type: Unknown		
Block size: $25 \times 25 \times 20$ ft		
Block weight w_i: variable		
zuck-large	96,821	30
Metal type: Unknown		
Block size: Unknown		
Block weight w_i: variable		
sm2	99,014	30
Metal type: Unknown		
Block size: Unknown		
Block weight w_i: variable		

This is a useful technique for problems with more constraints than variables [20]. All other CPLEX parameters were set to their default values. Note that although CPLEX is used in this study to solve the sub-problems (SP^t), any maximum flow algorithm can be used.

As explained in Sect. 3, the mining and processing capacities are multiplied by a factor α and by a factor β when solving the sub-problems (SP^t) and when applying the heuristic, respectively. α and β are random numbers in the intervals $[\alpha_1, \alpha_2]$ and $[\beta_1, \beta_2]$, and α_1, α_2, β_1, and β_2 are parameters of the solution procedure in the interval]0,1]. To fix the values of these parameters, some preliminary numerical experiments were completed with the largest two instances in each of the sets P_1 and P_2 (i.e., with instances C4, C5, G4, and G5). 15 different combinations generated with the 5 values for the interval $[\alpha_1, \alpha_2]$ ([0.7, 0.75],

Table 5 Evaluating the efficiency of **SH-VND**: instances generated from actual mineral deposits

Set	Problem	N	T	**SH-VND**				**CPLEX**
				$\%Min\ Gap$	$\%Max\ Gap$	$\%Ave\ Gap$	$Ave\ CPU$ (seconds)	$Ave\ CPU$ (seconds)
P_1	C1	4,273	3	0.63	0.91	0.74	0.49	11.38
	C2	7,141	4	0.72	0.79	0.74	1.51	145.36
	C3	12,627	7	1.28	2.31	1.95	5.10	2,250.15
	C4	20,626	10	1.63	2.82	2.11	14.89	21,919.40
	C5	26,021	13	1.50	1.86	1.66	23.87	47,237.90
P_2	G1	18,821	5	0.58	0.84	0.67	10.48	5,037.59
	G2	23,901	7	0.92	1.34	1.13	18.89	18,524.60
	G3	30,013	8	0.97	1.33	1.23	27.88	39,837.80
	G4	34,981	9	1.13	2.12	1.44	39.18	64,811.80
	G5	40,762	11	1.60	2.40	1.86	55.12	96,536.30

[0.65, 0.7], [0.6, 0.65], [0.55, 0.6], and [0.5, 0.55]) and the 3 values for the interval $[\beta_1, \beta_2]$ ([1, 1], [0.95, 1], and [0.9, 0.95]) were considered. The best results were obtained with the following combination:

- $[\alpha_1, \alpha_2] = [0.6, 0.65]$
- $[\beta_1, \beta_2] = [0.95, 1]$.

Hence, the rest of the numerical tests were completed using these values for the parameters for all instances.

Finally, the value of the parameter C, used to define the modified economic values $\bar{v_i}$ (i.e., the coefficients of the objective function (9)), was set to $C = N \max_i v_i$.

5.3 Hybrid method applied to instances generated from actual mineral deposits

The results of our computational experiments on the 10 instances described in Sect. 5.1.1 are reported below. We first present results for the variant of the MPSP described in Sect. 2 and commonly studied in the literature. Then we provide results for the variant where lower bounds on mining and processing are added to the constraint set. As mentioned Sect. 1, the variant incorporating lower bounds is considered in [12] and is known to be more difficult to solve.

5.3.1 MPSP described in Sect. 2

The linear relaxation of the MPSP formulation (1)–(8) was solved using the commercial solver CPLEX to obtain an upper bound on the optimal value, allowing us to assess the quality of the solutions produced with the proposed solution method, denoted **SH-VND** (**SH** stands for the sequential heuristic used to generate the initial solution and **VND** for the variable neighborhood descent used to improve the initial solution).

Since **SH-VND** includes random choices, each instance was solved 10 times. The results are summarized in Table 5. The first four columns give the name and the size of the instances. The next four columns display respectively

- $\%Min\ Gap = \frac{Z_{LR}-Z_{best}}{Z_{LR}} \times 100$: the value of the relative gap between the value Z_{best} of the best solution obtained by **SH-VND** over the 10 runs and the optimal value Z_{LR} of the linear relaxation.
- $\%Max\ Gap = \frac{Z_{LR}-Z_{worst}}{Z_{LR}} \times 100$: the value of the relative gap between the value Z_{worst} of the worst solution obtained by **SH-VND** over the 10 runs and the optimal value of the linear relaxation.
- $\%Ave\ Gap = \frac{Z_{LR}-Z_{average}}{Z_{LR}} \times 100$: the value of the relative gap between the average value $Z_{average}$ of the 10 solutions generated by **SH-VND** and the optimal value of the linear relaxation.
- $Ave\ CPU$: the average solution time in seconds.

The last column of Table 5 indicates the CPU time required by CPLEX to solve the linear relaxation of the problems. Note that the length of the interval $[\%Min\ Gap, \%Max\ Gap]$ indicates the robustness of the proposed method with respect to random choices. The following observations can be derived from Table 5:

- **SH-VND** is very efficient in the sense that for each problem the value of $\%Min\ Gap$ is less than 2 % away from the upper bound provided by CPLEX. This indicates that at least one of the 10 solutions generated is of excellent quality. Even though 10 runs are required to achieve this $\%Min\ Gap$, this is acceptable since the CPU time of a run is less than 1 min for all the tested instances.
- **SH-VND** is very robust in the sense that in all cases (considering the 10 instances and the 10 runs) the gap between the solution generated and the upper bound provided by CPLEX is smaller than 3 %, and in 92 % of all cases, it is smaller than 2 % (cf. values of $\%Max\ Gap$).
- The time required to find these high quality solutions is very reasonable. For the smallest problem, C1, a near-optimal solution is found almost immediately (less than 1 s). For the largest problem, G5, the CPU time is less than 1 min.
- As expected, the solution time increases with the number of blocks in the instance, but the rate of increase is larger for CPLEX than for **SH-VND**. Indeed, CPLEX may require more than 1 day (almost 27 h) to solve the linear relaxation of the largest problem, G5, while a very good feasible solution for this problem is obtained in less than 1 min by **SH-VND**.

Next the performance of **SH-VND** is investigated in more detail. Table 6 shows the improvement gained at each step of the algorithm. For each instance, we first give $\%Gap^{init}$, the gap of the initial solution computed as the relative difference between the value of this initial solution, obtained using the sequential heuristic (**SH**) described in Sect. 3, and the optimal value of the linear relaxation. This is followed by $\%Gap^{final}$, the gap of the final solution obtained after applying the **VND** procedure described in Sect. 4, and $\%Imp._{VND} = \frac{\%Gap^{init}-\%Gap^{final}}{\%Gap^{init}} \times 100$ as a measure of the improvement achieved by using **VND**. Note that the three measures above ($\%Gap^{init}$, $\%Gap^{final}$, and $\%Imp._{VND}$) are averaged over the 10 runs and hence $\%Gap^{final}$ corresponds to $\%Ave\ Gap$ in Table 5. We then report for each of the three neighborhood structures, used in our adaptation of **VND**, the two following measures also averaged over the 10 runs:

- $\#Times$: the number of times that the neighborhood has been explored
- $\%Imp._N$: the average improvement in percent gained by using the neighborhood. To compute the value of $\%Imp._N$, we proceed as follows: Suppose that the neighborhood is explored for the k^{th} time ($k = 1, \ldots, \#Times$). Let $\%Gap_k^{start}$ and $\%Gap_k^{end}$ be the

Table 6 Evaluating the efficiency of each step of **SH-VND**: instances generated from actual mineral deposits

Instance	$\%Gap^{init}$	$\%Gap^{final}$	$\%Imp._{VND}$	*Exchange* (N^1)		*Shift-after* (N^2)		*Shift-before* (N^3)	
				$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$
C1	1.29	0.74	42.61	5.40	4.82	4.00	0.28	3.20	7.05
C2	1.44	0.74	48.54	7.40	3.91	4.60	0.25	3.40	8.53
C3	3.01	1.95	35.20	8.70	2.15	7.10	0.18	4.20	6.42
C4	4.24	2.11	50.21	8.90	1.48	8.20	0.17	4.40	5.25
C5	3.65	1.66	54.54	9.90	1.37	8.70	0.58	5.10	9.48
G1	1.25	0.67	46.28	6.40	0.48	5.10	0.04	3.50	12.84
G2	1.96	1.13	42.18	8.00	0.57	6.40	0.04	3.50	11.19
G3	2.46	1.23	50.08	8.80	0.84	6.70	0.05	3.30	12.99
G4	3.31	1.44	56.29	9.20	0.94	6.60	0.10	4.00	13.83
G5	3.98	1.86	53.28	11.20	1.40	7.70	0.21	3.60	12.23

gap of the best solution found so far at the beginning and at the end of the exploration, respectively. The average improvement achieved by using the neighborhood is then:

$$\%Imp._N = \frac{\sum_{k=1}^{\#Times} \frac{\%Gap_k^{start} - \%Gap_k^{end}}{\%Gap_k^{start}} \times 100}{\#Times}.$$

From Table 6, it appears that **SH** was successful in determining good local optimal solutions for all tested instances and that **VND** was able to improve these initial solutions by up to 56 %. The three neighborhoods proved to perform well as they were all able to improve the solutions for all the tested instances (c.f. values of $\%Imp._N$). This clearly indicates that changing neighborhood is very useful for escaping local optima. The first neighborhood (*Exchange*) has been used more than the other two neighborhoods, and the second neighborhood (*Shift-after*) has been used more than the third one (*Shift-before*). This behaviour is due to the variable neighborhood search framework: Whenever a new local optimum is found, if it is better than the incumbent, the search is recentered around it and begins again with the first neighborhood; otherwise, the next neighborhood is used. Hence, the third neighborhood is only used if the first two neighborhoods fail to improve the best-known solution and the first neighborhood is used more frequently. Now, regarding the improvement gained by each neighborhood, one can see that the third neighborhood provides the best performance. We believe that the reason for such performance is that the extraction of many profitable blocks is advanced at each iteration when using this neighborhood, which increases the objective function considerably. Delaying the extraction of non-profitable blocks (the second neighborhood) has a modest impact as the blocks are selected in a greedy manner when generating the initial solution using **SH**, and thus many neighbor solutions are non-improving and are not selected. Finally, the first neighborhood has the two effects (delaying and advancing blocks), but only one block can be advanced at each iteration, and only moves that respect the *slope constraints* are allowed. This means that the third neighborhood is larger than the first neighborhood and thus provides better quality solutions. It is worth mentioning that if the value of the parameter β were fixed to 1 at each iteration of **SH**, then it would be more difficult to find a feasible neighbor solution when applying the *Shift-before* and *Shift-after* operators. This motivates

Table 7 Comparing **SH-VND** and **WM**

Set	Instance	N	T	$\%Ave\,Gap$			$Ave\,CPU$ (seconds)	
				SH-VND	**WM**	$\%Imp.$	**SH-VND**	**WM**
P_1	C1	4,273	3	0.74	14.48	94.89	0.49	43.00
	C2	7,141	4	0.74	18.26	95.95	1.51	64.00
	C3	12,627	7	1.95	19.58	90.04	5.10	78.00
	C4	20,626	10	2.11	20.98	89.94	14.89	85.00
	C5	26,021	13	1.66	3.19	47.96	23.87	101.00
P_2	G1	18,821	5	0.67	3.77	82.24	10.48	30.00
	G2	23,901	7	1.13	5.51	79.49	18.89	57.00
	G3	30,013	8	1.23	2.94	58.14	27.88	44.00
	G4	34,981	9	1.44	2.33	38.08	39.18	63.00
	G5	40,762	11	1.86	4.91	62.10	55.12	65.00

the introduction of constraints (12) and (13). This observation is corroborated by the results presented in Sect. 5.5.

To further assess the performance of **SH-VND**, we compare our results with those obtained by Whittle [28], which is, as mentioned earlier, a commercial mine planning software commonly used by professional mine planners. Three different algorithms are implemented in this software to solve the MPSP. We have used the Milawa NPV algorithm, which has been shown to find solutions with very high net present value (objective function (1)) compared to the other two algorithms [28]. In the following, we refer to the Whittle's Milawa NPV algorithm as **WM**.

The numerical results comparing the performance of **SH-VND** and **WM** on the 10 instances in sets P_1 and P_2 are summarized in Table 7. Each row is associated with an instance indicating the $\%Ave\,Gap$ and the $Ave\,CPU$ (as defined above) for each method. Note that each instance is solved only once with **WM** since the results of different runs are identical. We also give in Table 7 the value $\%Imp.$ defined as follows:

$$\%Imp. = \frac{\%Ave\,Gap \text{ with WM} - \%Ave\,Gap \text{ with SH}}{\%Ave\,Gap \text{ with WM}} \times 100$$

indicating the gain in reducing the $\%Ave\,Gap$ when using **SH-VND** instead of **WM**.

Clearly, **SH-VND** dominates **WM**. On average, when **SH-VND** is used, the $\%Ave\,Gap$ is improved by 74 %, and the $Ave\,CPU$ is reduced by a factor of 3. Moreover, it can be seen from the values of $\%Max\,Gap$ in Table 5 that, for all the tested instances, the 10 solutions generated by **SH-VND** are strictly better than the one produced by **WM**. On the other hand, from Table 7, it appears that the values of $\%Ave\,Gap$ obtained by **WM** are somewhat smaller for problems in P_2 than for problems in P_1, indicating that **WM** is not as robust as **SH-VND**, which gives similar results for all instances.

The results also indicate that, in general, **SH-VND** produces solutions with quite balanced mining and processing flows throughout the periods as compared with **WM**. Figure 5 shows illustrative results on the largest problem, G5.

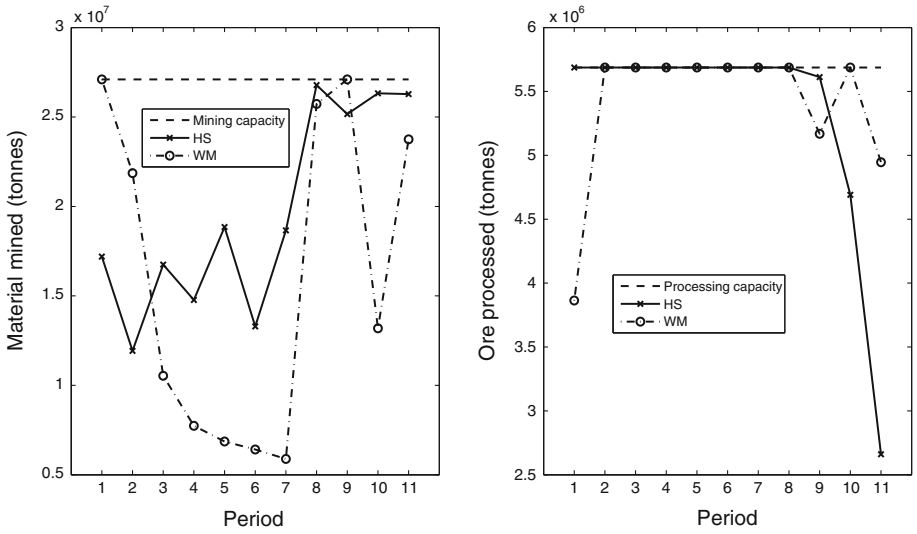

Fig. 5 Mining and processing flows throughout the periods for problem G5

5.3.2 MPSP with lower bounds on mining and processing

To avoid unbalanced mining and processing flows throughout the periods (i.e., situations like the ones depicted in Fig. 5), one can add to the MPSP formulation (1)–(8) the following set of constraints:

$$\sum_{i=1}^{N} w_i x_i^1 \geq \underline{M}^1 \tag{17}$$

$$\sum_{i=1}^{N} w_i (x_i^t - x_i^{t-1}) \geq \underline{M}^t \qquad t = 2, \ldots, T \tag{18}$$

$$\sum_{i=1}^{N} \theta_i w_i x_i^1 \geq \underline{\Theta}^1 \tag{19}$$

$$\sum_{i=1}^{N} \theta_i w_i (x_i^t - x_i^{t-1}) \geq \underline{\Theta}^t \qquad t = 2, \ldots, T. \tag{20}$$

Constraints (17) and (18) impose a lower bound $\underline{M}^t$ on the amount of material (*waste* and *ore*) mined during each period t. Constraints (19) and (20) ensure that the amount of *ore* processed during each period is at least equal to $\underline{\Theta}^t$.

As mentioned in [12], these constraints are important for practical applications as they can prevent large set-up costs incurred by stopping and restarting the processing operations. However, including them adds significantly to the complexity of the problem and hampers certain specialized computational methods proposed in the literature. The authors note that "even a small problem with lower bounds on resource consumption can be dramatically harder to solve than the same problem with those lower bounds omitted." They give an example of a small test problem solved with CPLEX for which the solution time increases by a factor of 21 when lower bounds are added. To further test the proposed **SH-VND**

Table 8 Evaluating the efficiency of **SH-VND** to account for additional constraints: instances generated from actual mineral deposits

Set	Problem	N	T	**SH-VND**				**CPLEX**
				$\%Min\ Gap$	$\%Max\ Gap$	$\%Ave\ Gap$	$Ave\ CPU$ (s)	$Ave\ CPU$ (s)
P_1	C1	4,273	3	0.63	0.91	0.74	0.49	10.72
	C2	7,141	4	0.64	0.72	0.68	1.55	72.5
	C3	12,627	7	1.18	2.36	1.93	15.55	3,019.79
	C4	20,626	10	1.32	3.51	2.33	129.89	32,639.20
	C5	26,021	13	2.77	3.91	3.45	1,723.22	143,676.00
P_2	G1	18,821	5	0.58	0.84	0.67	11.64	3,118.11
	G2	23,901	7	0.94	1.33	1.13	20.80	19,430.60
	G3	30,013	8	0.98	2.01	1.29	34.50	35,087.70
	G4	34,981	9	1.01	2.23	1.70	51.02	63,296.70
	G5	40,762	11	2.19	3.54	2.92	1,108.80	146,317.00

method, we examine how it performs when lower bounds on mining and processing are added (constraints (17)–(20)). Accounting for these constraints requires minor modifications to the solution procedure, which are described below.

Consider an initial solution generated using the procedure described in Sect. 3 and with the same parameter values as in Sect. 5.2. A repair method based on a simple tabu search is applied if this solution does not satisfy constraints (17)–(20). The neighborhood is obtained by moving a block i currently scheduled at period t to another period $\tau \neq t$ provided that the *slope constraints* are satisfied. When moving i from t to τ, block i is forbidden to be scheduled at t for the next $numIter$ iterations, where $numIter$ is a random integer number chosen in $[0.8N, 1.2N]$ (moving i to t is declared tabu). At each iteration, one of the best non-tabu neighbor solutions is selected, and the tabu search terminates when the solution is feasible. To maintain the feasibility of the solution during the second phase of the solution procedure (where we apply variable neighborhood descent), when exploring any of the three neighborhoods (*Exchange, Shift-after, and Shift-before*), moves that would violate constraints (17)–(20) are not allowed.

Table 8 shows the results of applying the modified version of **SH-VND** to the 10 test instances described in Sect. 5.1.1. Lower bounds on mining $\underline{W}^t$ are set to $\lceil 0.75 \frac{\sum_{i=1}^{N} w_i}{T} \rceil$ and $\lceil 0.70 \frac{\sum_{i=1}^{N} w_i}{T} \rceil$ for instances in P_1 and P_2, respectively (i.e., $\frac{\text{total amount of rock}}{\text{number of periods}}$ minus a margin of 25 % and 30 %, respectively). Lower bounds on processing $\underline{\Theta}^t$ are set to $\lceil 0.75 \frac{\sum_{i=1}^{N} w_i \theta_i}{T} \rceil$ (i.e, $\frac{\text{total amount of } ore}{\text{number of periods}}$ minus a margin of 25 %). We solved the linear relaxation of the MPSP formulation (1)–(8),(17)–(20) using version 12.5 of the commercial solver CPLEX to obtain an upper bound on the optimal value, and we use the same criteria as in Sect. 5.3.1 to evaluate the efficiency of the modified version of **SH-VND** to deal with the variant of the MPSP with additional operational constraints (lower bounds on mining and processing).

As can be seen, globally, **SH-VND** still provides very good quality solutions in reasonable computational times when lower bounds on mining and processing are accounted for. The average gap is only 1.68 % (versus 1.35 % when lower bounds are omitted). The small difference between the values of $\%Min\ Gap$ and $\%Max\ Gap$ indicates the robustness of

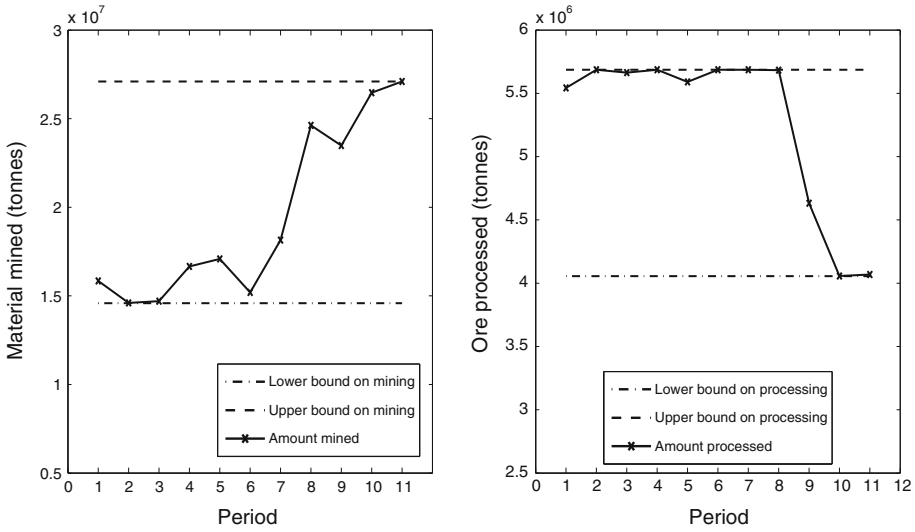

Fig. 6 Mining and processing flows throughout the periods for problem G5 when lower bounds are added

SH-VND. Overall, solution times tend to remain reasonable. We observe that for instances C1, C2, G1, and G2, adding lower bounds does not affect the solution time (CPU times are comparable to those when lower bounds are omitted, c.f. Table 5). For these instances, the initial solutions generated were actually in most of the runs feasible, so the repair heuristic (tabu search) was not necessary. However, for the other instances, the solutions times are longer because it is hard to find a feasible solution. For the largest problem in the set P_1, C5, the bounds are very tight, and typically about 80 % of the CPU time is spent in looking for a feasible solution. Although the solution times required by **SH-VND** are longer when we account for the lower bounds constraints than when we don't (range between 1 s and 29 min versus 1 s and 1 min), they are still significantly smaller than those required by CPLEX to solve the linear relaxation of the problem. For instance, for the largest instance G5, CPLEX requires almost 2 days to solve the linear relaxation of the problem, while **SH-VND** can find a near-optimal solution in only about 18 min on average. Figure 6 shows the mining and processing flows throughout the periods for this problem in a solution generated by **SH-VND**.

Next, Table 9 presents detailed results allowing us to analyze the efficiency of each step of the proposed algorithm. This table has the same structure as Table 6. Similar conclusions, as in solving the variant without lower bounds on mining and processing, are obtained:

- For each instance, except C5, the initial solutions are of very good quality, within less than 5 % of optimality
- **VND** was able to improve the initial solutions by more than 50 % on average
- The three neighborhoods perform well as they all improve the solutions for all the tested instances
- The *Shift-before* (N^2) neighborhood is again the best one; that is, the one that improves the most the solution
- The *Shift-after* (N^3) neighborhood is not competitive although it performs slightly better than when solving the variant where the lower bounds are omitted, especially for the largest problems, C5 and G5, which are much harder to solve.

Table 9 Evaluating the efficiency of each step of **SH-VND** when lower bounds on mining and processing are added: instances generated from actual mineral deposits

Instance	$\%Gap^{init}$	$\%Gap^{final}$	$\%Imp._{VND}$	*Exchange* (N^1)		*Shift-after* (N^2)		*Shift-before* (N^3)	
				$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$
C1	1.29	0.74	42.61	5.40	4.82	4.00	0.28	3.20	7.05
C2	1.32	0.68	48.42	7.70	3.69	5.00	0.17	3.40	8.59
C3	2.94	1.93	34.28	20.80	1.05	18.50	0.25	3.60	3.77
C4	4.72	2.33	50.64	14.80	1.14	16.00	0.24	4.90	2.89
C5	10.31	3.45	66.54	21.00	0.69	22.50	2.23	7.70	1.92
G1	1.25	0.67	46.28	6.40	0.48	5.10	0.04	3.50	12.84
G2	1.91	1.13	40.88	8.10	0.58	7.10	0.05	3.70	10.56
G3	2.41	1.29	46.43	9.60	0.85	9.70	0.09	4.50	8.84
G4	3.26	1.70	47.84	12.20	0.85	9.50	0.11	7.20	6.96
G5	4.60	2.92	36.59	18.30	0.46	19.90	1.15	4.20	2.02

Notice also that when solving the variant of MPSP without lower bounds on mining and processing, **VND** performs fewer iterations than when solving the variant with lower bounds (c.f. columns $\#Times$ in Tables 6 and 9). This partly explains the differences in CPU times (c.f. columns $AveCPU$ in Tables 5 and 8).

5.4 Hybrid method applied to the instances in the first set of benchmark instances

We now report the results of our computational experiments on the two benchmark instances described in Sect. 5.1.2. We first present results for the case where only upper bounds on mining and processing are considered, and then we provide results for the variant of the problem where lower bounds on mining and processing are considered as well.

5.4.1 MPSP described in Sect. 2

The results produced by the proposed method, **SH-VND**, are compared with those produced by the method described in [11] and summarized in Sect. 1, from now on **ExTS-LS**. We do not compare **SH-VND** with the methods in [1] and [22] because they have been shown to be dominated by **ExTS-LS**. In Table 10, we use the same comparison criteria as in the previous section; i.e., $\%MinGap$, $\%MaxGap$, $\%AveGap$, and $AveCPU$. The values of the gaps and the CPU times for **ExTS-LS** have been taken from [11]. Note that the values of $\%MinGap$, $\%MaxGap$, and $\%AveGap$ are identical for this method since it provides only one solution. Furthermore, in [11], to evaluate the quality of the solutions, the authors use the value of the solution generated by their method divided by the optimal value of the linear relaxation rather than the value of the relative gap between the value of the solution generated and the optimal value of the linear relaxation. Therefore, in the interest of consistency with the results presented in the previous sections, we convert the values of the criterion they use to the relative gap.

From Table 10, it can be seen that for the Marvin instance **SH-VND** significantly dominates **ExTS-LS**. On average, the gap is improved by 17 % and the solution time is reduced by a factor of 450 when **SH-VND** is used. Furthermore, all the solutions generated by **SH-VND** over the 10 runs are better than the one found using **ExTS-LS**. Although **SH-VND** is not as

Table 10 Comparing **SH-VND** and **ExTS-LS**: instances in the first benchmark set

Instance	N	T	$\%Min\ Gap$		$\%Max\ Gap$		$\%Ave\ Gap$		$Ave\ CPU$ (s)	
			SH-VND	**ExTS-LS**	**SH-VND**	**ExTS-LS**	**SH-VND**	**ExTS-LS**	**SH-VND**	**ExTS-LS**
AmericaMine	19,320	15	3.56	1.00	4.43	1.00	3.96	1.00	17.71	28, 800.00
Marvin	53,668	15	1.71	2.30	2.12	2.30	1.90	2.30	64.06	28, 800.00

Table 11 Evaluating the efficiency of each step of **SH-VND**: instances in the first benchmark set

Instance	$\%Gap^{init}$	$\%Gap^{final}$	$\%Imp._{VND}$	*Exchange* (N^1)		*Shift-after* (N^2)		*Shift-before* (N^3)	
				$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$
AmericaMine	6.25	3.96	36.64	15.50	1.31	12.20	0.50	5.90	3.50
Marvin	4.19	1.90	54.69	14.40	1.66	10.50	0.02	5.00	8.38

Table 12 Evaluating the efficiency of **SH-VND** to account for additional constraints: instances in the first benchmark set

Problem	N	T	**SH-VND**				**CPLEX**
			$\%Min\ Gap$	$\%Max\ Gap$	$\%Ave\ Gap$	$Ave\ CPU$ (s)	$Ave\ CPU$ (s)
AmericaMine	19,320	15	3.91	4.80	4.40	18.21	2,624.42
Marvin	53,668	15	3.99	5.34	4.63	67.00	97,702.30

successful as **ExTS-LS** on the AmericaMine instance, it is still robust and able to find good solutions quickly. In [11], the authors report that the gap obtained after 15 min of running their local search procedure is about 4.50 %, indicating that the 10 solutions we found in less than 18 s are on average 12 % better than **ExTS-LS** solution after 15 min. Note that 15 min represent the time spent to improve the solution and doesn't include the time required to generate the initial solution.

The question addressed next is whether **SH** and the three neighborhoods exhibit the same performance as when solving the 10 instances generated from actual mineral deposits (instances C1-C5 and G1-G5); that is, if the initial solution is of good quality, if VND is able to improve it, if the three neighborhoods are used, and if the *Shift-before* (N^3) neighborhood outperforms the other two neighborhoods. The answer is positive as can be seen from the results in Table 11. This table has the same structure as Tables 6 and 9.

5.4.2 MPSP with lower bounds on mining and processing

The sliding time window heuristic (**STWH**) proposed in [12] to solve the variant of MPSP with lower bounds on mining and processing is limited by the size of the instances. Indeed, the authors couldn't solve the Marvin instance and solved instead "slices" or parts of it with up to 25,000 blocks. The range of the gap in their results is between 1.4 % for the smallest instance and 4.3 % for the largest one, and the solution times are between 30 min and 2.7 h. Results presented in Table 12 clearly show the superiority of the method proposed in this paper, **SH-VND**, over **STWH**. **SH-VND** is able to tackle Marvin, and provides in almost

Table 13 Evaluating the efficiency of each step of **SH-VND** when solving the instances in the first benchmark set with lower bounds on mining and processing

Instance	$\%Gap^{init}$	$\%Gap^{final}$	$\%Imp._{VND}$	*Exchange* (N^1)		*Shift-after* (N^2)		*Shift-before* (N^3)	
				$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$
AmericaMine	6.47	4.40	32.05	14.40	1.03	13.70	0.47	5.70	2.71
Marvin	5.78	4.63	19.75	17.30	0.83	17.80	0.02	4.90	1.44

Table 14 Testing **SH-VND** on instances from *MineLib*

Instance	N	T	$\%Ave\ Gap$ **SH-VND**	*Best Known Gap* **LR+Modified TopoSort**	$Ave\ CPU$ (s) **SH-VND**
newman1	1,060	6	**1.68**	4.10	0.08
zuck-small	9,400	20	**1.80**	7.70	7.77
zuck-medium	29,277	15	**8.08**	13.40	121.60
p4hd	40,947	10	6.61	**0.50**	78.57
marvin	53,271	20	**1.87**	5.00	70.74
w23	74,260	12	9.96	**2.10**	203.46
zuck-large	96,821	30	8.07	**1.10**	1,003.97
sm2	99,014	30	5.85	**0.20**	105.00

1 min very good solutions with an average gap of 4.63 %. Regarding the efficiency of the steps of **SH-VND**, results reported in Table 13 indicate that each step of the algorithm performs similarly as in the experiments discussed in the previous sections: good initial solutions with **SH**, improved significantly by **VND**, and the largest improvement are obtained with the third neighborhood, *Shift-before*.

5.5 Hybrid method applied to the instances in the second set of benchmark instances (instances from *MineLib*)

As mentioned earlier in Sect. 5.1.3, the method proposed in this paper, **SH-VND**, was also compared on instances from *MineLib*. The results are given in Table 14. We first recall in the first three columns the name of the instances and their sizes. Then, we give the $\%Ave\ Gap$, as defined earlier (the value of the relative gap between the average value $Z_{average}$ of the 10 solutions generated by **SH-VND** and the optimal value of the linear relaxation). This time, the optimal value of the linear relaxation has not been obtained using CPLEX, but it has been taken from [15]. The authors mention that it has been computed using a modified version of Bienstock-Zuckerberg's algorithm [3]. The criterion $\%Ave\ Gap$ is followed by $Best\ Known\ Gap$, the value of the gap of the current best-known integer-feasible solution computed as the relative difference between the value of this feasible solution and the optimal value of the linear relaxation. The values of $Best\ Known\ Gap$ are also taken from [15]. The authors report that the current best-known integer-feasible solution has been obtained from the LP relaxation using a modified version of the TopoSort heuristic. The CPU time (in seconds) required by **SH-VND** to solve each instance is given in the last column of the table. We do not report the CPU times of the method used to obtain the current best-known solutions because they are provided neither in [15] nor in http://mansci.uai.cl/minelib. Finally, when comparing

Table 15 Evaluating the efficiency of each step of **SH-VND** when solving the instances in the second benchmark set (instances from *MineLib*)

Instance	$\%Gap^{init}$	$\%Gap^{final}$	$\%Imp._{VND}$	*Exchange* (N^1)		*Shift-after* (N^2)		*Shift-before* (N^3)	
				$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$
newman1	2.58	1.68	34.89	8.90	2.53	4.10	0.01	7.50	2.55
zuck-small	3.83	1.80	53.10	14.70	1.70	10.50	0.02	7.90	5.07
zuck-medium	10.20	8.08	20.80	21.00	0.58	11.80	0.07	6.70	1.54
p4hd	10.08	6.61	34.40	23.50	1.07	17.90	0.10	9.20	1.49
marvin	3.79	1.87	50.78	14.50	1.35	12.80	0.02	8.60	4.64
w23	15.54	9.96	35.91	33.70	0.52	21.70	0.04	15.40	1.68
zuck-large	13.28	8.07	39.21	53.20	0.75	28.00	0.07	9.70	0.94
sm2	8.13	5.85	27.99	48.20	0.19	28.40	0.09	57.10	0.37

$\%Ave\ Gap$ with *Best Known Gap* the best results are indicated with bold numbers. Results evaluating the efficiency of each step of **SH-VND** when solving the instances from *MineLib* are summarized in Table 15, which has the same structure as Tables 6, 9, 11, and 13.

The results in Table 14 are promising as they indicate that the best-known solution was improved for 4 out of the 8 instances from *MineLib*. The computational time required for solving each of the 8 instances is comparable with the time required to solve the instances from the other sets (previous experiments); that is, it ranges from a fraction of second to few minutes, even for the largest instances with more than 95,000 blocks and 30 periods. Regarding the efficiency of each step of **SH-VND** (c.f. Table 15), the results are also similar to the previous experiments with the other sets of instances: **VND** is efficient in improving the initial solutions generated by **SH**, and overall, the *Shift-before* neighborhood (N^3) would rank first, the *Exchange* neighborhood (N^1) second, and the *Shift-after* neighborhood (N^2) last.

To further analyze the performance of the proposed variable neighborhood descent procedure (**VND**), additional tests were performed on the same 8 instances from *MineLib*, but instead of starting **VND** from the solution generated using **SH**, it is started using the current best-known integer-feasible solution provided in http://mansci.uai.cl/minelib. Recall that this solution has been obtained from the LP relaxation using a modified version of the TopoSort heuristic (henceforth referred to as **LR-MTS**). The $\%Ave\ Gap$ obtained using this variant of the solution procedure, denoted in what follows **LR-MTS-VND**, is given in Table 16. We also recall in this table the values of $\%Ave\ Gap$ obtained using **SH-VND** and **LR-MTS**. Bold numbers indicate the best results. The criterion $Ave\ CPU$ (solution time) is not used in the comparison because, as mentioned earlier, we don't know the CPU times of **LR-MTS**. Finally, details about the efficiency of each step of **LR-MTS-VND** are summarized in Table 17.

Notice first that for each instance, we have been able to improve on **LR-MTS** results; that is, the best results previously published, by using the **VND** approach either combined with **SH** or with **LR-MTS**. As can be observed, **LR-MTS-VND** outperforms **LR-MTS**, as the best-known solution to each instance has been improved using **LR-MTS-VND**. However, **LR-MTS-VND** does not always produce better results than **SH-VND** (4 out of the 8 instances only). Regarding the improvement in percent gained by **VND** ($\%Imp._{VND}$), it can be seen from Tables 15 and 17 that, on average, $\%Imp._{VND}$ is higher when the initial solution

Table 16 Comparing **SH-VND**, **LR-MTS**, and **LR-MTS-VND** on instances from *MineLib*

Instance	$\%Ave\,Gap$		
	SH-VND	**LR-MTS**	**LR-MTS-VND**
newman1	**1.68**	4.10	2.34
zuck-small	**1.80**	7.70	4.88
zuck-medium	**8.08**	13.40	9.04
p4hd	6.61	0.50	**0.19**
marvin	**1.87**	5.00	3.20
w23	9.96	2.10	**1.06**
zuck-large	8.07	1.10	**0.56**
sm2	5.85	0.20	**0.04**

Table 17 Evaluating the efficiency of each step of **LR-MTS-VND** when solving the instances in the second benchmark set (instances from *MineLib*)

Instance	$\%Gap^{init}$	$\%Gap^{final}$	$\%Imp._{VND}$	*Exchange* (N^1)		*Shift-after* (N^2)		*Shift-before* (N^3)	
				$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$	$\#Times$	$\%Imp._N$
newman1	4.09	2.34	42.85	10.90	3.97	6.30	0.01	5.00	1.65
zuck-small	7.67	4.88	36.39	19.20	1.73	17.70	0.59	6.30	0.07
zuck-medium	13.40	9.04	32.56	31.30	0.50	23.20	1.02	4.20	0.05
p4hd	0.52	0.19	62.46	28.10	2.15	25.10	1.33	3.90	0.51
marvin	5.00	3.20	35.95	13.00	1.65	11.00	1.93	1.00	0.00
w23	2.10	1.06	49.45	15.30	1.33	11.40	3.49	3.90	0.02
zuck-large	1.07	0.56	47.35	18.60	1.12	14.10	2.72	3.70	0.01
sm2	0.17	0.04	78.77	20.50	5.44	10.00	0.00	16.00	0.04

is generated with **LR-MTS** than when it is generated with **SH** (48.22 % versus 37.13 %). Finally, when considering the efficiency of each neighborhood, one can observe that the efficiency of the neighborhood *Shift-before* is reduced when the initial solution is generated using **LR-MTS** (recall that in all previous experiments, this neighborhood was the most efficient one). This is due to the fact that in the initial solutions generated using **LR-MTS**, either the mining capacity or the processing capacity is reached in almost all periods where blocks are mined. This restricts the efficiency of the *Shift-before* neighborhood since it is impossible to find a feasible neighbor solution when applying this operator in most periods. Table 17 shows that when this situation occurs, the *Exchange* neighborhood is the best one among the three neighborhoods considered in this paper. These results clearly confirm that the use of more than one neighborhood is very useful for escaping local optima.

5.6 Summary

The proposed solution method **SH-VND** outperforms CPLEX in terms of solution time. It is worth mentioning though that, unlike CPLEX, it doesn't provide optimal solutions but rather sub-optimal solutions (very close to the optimal for almost all the tested instances). **SH-VND** outperforms Whittle, the commercial mine planning software commonly used by professional mine planners, in terms of efficiency and robustness. The 20 instances considered

in this paper were solved within less than 3.2 % of optimality, on average, in less than 2 min. When comparing **SH-VND** to recent solution methods proposed in the literature, the results indicate that it is better than the **ExTS-LS** method proposed by [11], providing an excellent compromise between solution time and solution quality, as well as a new best-known solution for the instance Marvin. **SH-VND** and/or **LR-MTS-VND** also provide new best-known solutions for all the 8 instances from *MineLib* [15], indicating their superiority over the **LR-MTS** method. Another interesting feature of the proposed solution method is that it is not a specialized method tailored to solve one variant of the open-pit mine production scheduling problem, but it can be easily adapted to account for additional operational constraints. We have modified it to account for lower bounds on mining and processing. The results indicate that although the average CPU time is higher than in the case when the lower bounds are omitted, the solution quality does not deteriorate when accounting for lower bounds. On average, considering the 12 instances in which lower bounds on mining and processing were added, the gap is slightly above 2 % (2.15 %) and the solution time is only about 4 min. The Marvin instance is intractable by the method proposed in [12], but it is successfully solved by the method we have developed, **SH-VND**, indicating the superiority of **SH-VND** over that method. Because **SH-VND** doesn't rely on solving integer programming problems, its running time has a slower growth rate compared to other solution methods in the literature such as the ones in [11] and [12]. Indeed, the CPU time of **SH-VND** ranges between a fraction of second and 16 min for instances whose sizes vary from 1,060 blocks and 6 years to 99,014 blocks and 30 periods.

6 Conclusions

Production scheduling is a challenging and critical issue for mining companies exploiting open-pit mines. Determining the block mining sequence is a crucial step in maximizing the net present value of the mining operation. It involves significant capital investment in the order of hundreds of millions of dollars and is a key factor in determining investment returns. Decisions on block scheduling are subject to various types of constraints, typically slope constraints, bounds on mining, and bounds on processing. An additional characteristic of open-pit mine production scheduling, which makes the problem even more difficult, is that the number of blocks is large, in the order of tens to hundreds of thousands, yielding a large-scale optimization problem.

We have proposed a hybrid method (**SH-VND**), based on linear programming and variable neighborhood descent, able to solve large instances of this problem in a short amount of time. Unlike recent solution methods, **SH-VND** does not rely on time-consuming integer programming algorithms, and we believe this is one reason for its success. Instead, it relies on solving a series of linear programs to generate an initial solution. A variable neighborhood descent procedure is then applied to improve this solution.

Upper bounds provided by CPLEX were used to evaluate the efficiency of the proposed solution method. The performance of the method was also assessed by comparing it to recent solution methods proposed in the literature and to an alternate method implemented in commercial mine planning software commonly used by professional mine planners. The results indicate that **SH-VND** is superior to existing solution approaches, allowing us to find high quality solutions in very short computational times. The average quality of the solutions produced overall is better than previously published results.

SH-VND can also easily handle more complex variants of the open-pit mine production scheduling problem, and the computational experiments indicate that it is successful on the

variant incorporating lower bounds on mining and processing, able to find very good solutions for instances intractable with recently-published algorithms. Another important feature of the proposed method is that it doesn't require any external software, such as CPLEX, in order to be implemented. Indeed, although CPLEX is used in this study to solve the sub-problems when generating the initial solution, any maximum flow algorithm can be used.

Future research will be devoted to extending the method to account for uncertainty in the metal content of blocks and in metal prices.

Acknowledgments The work in this paper was funded by NSERC CRDPJ 411270-10, NSERC Discovery Grant 239019-06, and the industry members of the COSMO Stochastic Mine Planning Laboratory: AngloGold Ashanti, Barrick, BHP Billiton, De Beers, Newmont, and Vale. Thanks are in order to Brian Baird, Peter Stone, Darren Dyck, and Gavin Yates of BHP Billiton for their support, collaboration, and technical comments. The authors also thank two anonymous referees for their valuable comments and suggestions that helped improve the paper.

References

1. Amaya, J., Espinoza, D., Goycoolea, M., Moreno, E., Prevost, T., Rubio, E.: A scalable approach to optimal block scheduling. In: Proceedings of APCOM, pp. 567–575 (2009)
2. Asad, M.W.A., Dimitrakopoulos, R.: Implementing a parametric maximum flow algorithm for optimum open pit mine design under uncertain supply and demand. J. Oper. Res. Soc. **64**, 185–197 (2013)
3. Bienstock, D., Zuckerberg, M.: Solving LP relaxations of large-scale precedence constrained problems. Lect. Notes Comput. Sci. **6080**, 1–14 (2010)
4. Bley, A., Boland, N., Fricke, C., Froyland, G.: A strengthened formulation and cutting planes for the open pit mine production scheduling problem. Comput. Oper. Res. **37**, 1641–1647 (2010)
5. Boland, N., Dumitrescu, I., Froyland, G., Gleixner, A.M.: LP-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. Comput. Oper. Res. **36**, 1064–1089 (2009)
6. Boland, N., Dumitrescu, I., Froyland, G.: A multistage stochastic programming approach to open pit mine production scheduling with uncertain geology, Optimization Online. http://www.optimization-online.org/DB_FILE/2008/10/2123.pdf. Accessed 21 Oct 2013 (2008)
7. Lamghari, A., Dimitrakopoulos, R.: A diversified tabu search approach for the open-pit mine production scheduling problem with metal uncertainty. Eur. J. Oper. Res. **212**, 642–652 (2012)
8. Brimberg, J., Hansen, P., Mladenovic, N., Taillard, E.: Improvements and comparison of heuristics for solving the multisource Weber problem. Oper. Res. **48**, 444–460 (2000)
9. Caccetta, L., Hill, S.P.: An application of branch and cut to open pit mine scheduling. J. Glob. Optim. **27**, 349–365 (2003)
10. Chatterjee, S., Lamghari, A., Dimitrakopoulos, R.: A two-phase heuristic method for constrained pushback design. In: Castro, R., Emery, X., Kuyvenhoven, R. (eds.) Proceedings of MININ 2010, Conference on Mining Innovation. Santiago, Chile, pp. 195–204 (2010)
11. Chicoisne, R., Espinoza, D., Goycoolea, M., Moreno, E., Rubio, E.: A new algorithm for the open-pit mine production scheduling problem. Oper. Res. **60**, 517–528 (2012)
12. Cullenbine, C., Wood, R., Newman, A.: A sliding time window heuristic for open pit mine block sequencing. Optim. Lett. **5**, 365–377 (2011)
13. Dagdelen, K., Johnson, T.B.: Optimum open pit mine production scheduling by lagrangian parameterization. In: Proceedings of 19th International APCOM Symposium, pp. 127–142. Littleton, CO (1986)
14. Denby, B., Schofield, D.: Open-pit design and scheduling by use of genetic algorithms. Trans. Inst. Min. Metall. **103**, A21–A26 (1994)
15. Espinoza, D., Goycoolea, M., Moreno, E., Newman, A.: MineLib: a library of open pit mining problems. Ann. Oper. Res. **206**, 93–114 (2013)
16. Ferland, J.A., Amaya, J., Djuimo, M.S.: Application of a particle swarm algorithm to the capacitated open pit mining problem. In: Mukhopadhyay, S., Sen Gupta, G. (eds.) Autonomous Robots and Agents. Springer, Berlin (2007)

17. Gershon, M.: Heuristic approaches for mine planning and production scheduling. Int. J. Min. Geol. Eng. **5**, 1–13 (1987)
18. Hansen, P., Mladenovic, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–467 (2001)
19. Hochbaum, D.C.: The pseudoflow algorithm: a new algorithm for the maximum-flow problem. Oper. Res. **56**, 992–1009 (2008)
20. IBM: IBM ILOG AMPL version 12.2 users guide: standard (command-line) version including CPLEX directives (2010)
21. Johnson, T.: Optimum production scheduling. In: Proceedings of the 8th international symposium on computers and operations research, pp. 539–562. Salt Lake City (1969)
22. Moreno, E., Espinoza, D., Goycoolea, M.: Large-scale multi-period precedence constrained knapsack problem: a mining application. Electron. Notes Discret. Math. **36**, 407–414 (2010)
23. Newman, A.M., Rubio, E., Caro, R., Weintraub, A., Eurek, K.: A review of operations research in mine planning. Interfaces **40**, 222–245 (2010)
24. Ramazan, S.: The new fundamental tree algorithm for production scheduling of open pit mines. Eur. J. Oper. Res. **177**, 1153–1166 (2007)
25. Ramazan, S., Dimitrakopoulos, R.: Recent applications of operations research in open pit mining. Trans. Soc. Min. Metall. Explor. **316**, 73–78 (2004)
26. Sevim, H., Lei, D.: The problem of production planning in open pit mines. INFOR **36**, 1–12 (1998)
27. Tolwinski, B., Underwood, R.: A scheduling algorithm for open pit mines. IMA J. Math. Appl. Bus. Ind. **7**, 247–270 (1996)
28. Whittle, J.: Four-X User Manual. Whittle Programming Pty Ltd., Melbourne (1998)