Les Cahiers du GERAD ISSN: 0711–2440

Hyper-heuristic approaches for solving stochastic optimization formulations of mineral value chains

A. Lamghari,
R. Dimitrakopoulos

G–2016–105

November 2016

Cette version est mise à votre disposition conformément à la politique de libre accès aux publications des organismes subventionnaires canadiens et québécois.

Avant de citer ce rapport, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2016-105) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

This version is available to you under the open access policy of Canadian and Quebec funding agencies.

Before citing this report, please visit our website (https://www.gerad.ca/en/papers/G-2016-105) to update your reference data, if it has been published in a scientific journal.

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2016
– Bibliothèque et Archives Canada, 2016

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*.

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2016
– Library and Archives Canada, 2016

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Hyper-heuristic approaches for solving stochastic optimization formulations of mineral value chains

Amina Lamghari [a]

Roussos Dimitrakopoulos [a]

[a] *GERAD & COSMO Stochastic Mine Planning Laboratory, Department of Mining and Materials Engineering, McGill University, FDA Building, 3450 University Street, Montreal, Quebec, H3A 0E8, Canada*

amina.lamghari@mcgill.ca
roussos.dimitrakopoulos@mcgill.ca

November 2016

Les Cahiers du GERAD

G–2016–105

Copyright © 2016 GERAD

Abstract: This paper presents three hyper-heuristic approaches for the stochastic open-pit mine production scheduling problem with one processing stream (SMPS) and one of its generalizations, SMPS with multiple processing streams and stockpiles (SMPS+), which aim to optimize the associated mineral value chains. Two of the proposed hyper-heuristic approaches are refined versions of approaches that have been previously proposed in the literature and applied to solve other optimization problems, while the third one is a novel approach that uses some of the ideas of the first two but also includes new features aimed to overcome their weaknesses. The three approaches are simple, fast, and general. Their performance is assessed by comparing them to each other and to other search methodologies from the literature on benchmark instances of various sizes and characteristics. This comparison indicates that the new proposed hyper-heuristic outperforms the two others, providing results that are comparable to or improve on the results obtained by the state-of-the-art problem-specific methods.

Keywords: Hyper-heuristics, mineral value chains, strategic planning, uncertainty, local search.

1 Introduction

A mineral value chain is an integrated business that extracts materials from open-pit mines and/or underground mines, treats the extracted materials using a set of processing facilities linked via different handling methods, and generates a set of products that are sold to customers or on the spot market (Goodfellow and Dimitrakopoulos, 2014). When optimizing a mineral value chain, the goal is to generate a long-term production plan that maximizes the net present value (NPV) of the chain while meeting various physical and operational requirements at the extraction and processing levels. These strategic level decisions affect the subsequent tactical decisions and are a key factor in determining the profitability and efficiency of a mining operation, which involves huge capital investments.

Every mining operation has its own characteristics that may vary widely from one operation to another. The sources of supply might be open-pit mines, underground mines, or both. The processing facilities and the processing paths might also be different, as they depend on the main minerals in the mines, on the different products produced in the various processing streams, on the commodity produced, and on the geographical location. Hence, there are different mineral value chains, and the type of a mineral value chain depends on its components. The simplest and most commonly studied one in the literature consists of one open-pit mine, one processing facility, and one waste dump. The associated optimization problem is often referred to as the open-pit mine production scheduling problem (MPS). A planning horizon of T periods is considered. The mine is discretized into a set of N blocks, each of which represents a volume of material that can be mined. With each block i are associated a weight w_i, a metal content m_i, and a set of predecessors $P(i)$ representing the set of blocks that have to be mined in order to have access to i. The MPS consists of designing a mining sequence such that: (i) every block is mined at most once; (ii) every block is mined after its predecessors; (iii) the total amount of material mined in period t does not exceed W^t, the extraction capacity; (iv) the total amount of ore processed in period t does not exceed Θ^t, the processing capacity; and (v) the NPV of the mining operation is maximized. MPS is modelled as a linear integer program. It generalizes the constrained maximum closure problem and is therefore NP-hard (Hochbaum and Chen, 2000; Bienstock and Zuckerberg, 2010). This makes solving large instances of practical interest computationally challenging and beyond the scope of exact methods and general-purpose solvers.[1] This difficulty is exacerbated by the need to incorporate additional operational constraints. For example, lower bounds on mining, processing, and metal production are often required to ensure that the resources are utilized evenly and that the demand is satisfied at each period. While this is useful and closer to the problem encountered in practical settings, the introduction of such constraints in the MPS formulation makes the problem harder to solve (Cullenbine et al., 2011). Another realistic and important aspect that further complicates the solution process is metal uncertainty. Metal uncertainty, also referred to as geological or reserve uncertainty, stems from the fact that the metal content of the blocks is not known at the time decisions are made, but it is inferred from limited drilling data. The benefits of integrating metal uncertainty in the optimization process are well-documented in the literature. By taking into account the effects of metal uncertainty on present decision-making, not only is risk in meeting production targets reduced, but also major improvements in NPV in the order of 10 to 30% are reached (Ravenscroft, 1992; Dowd, 1994; Dimitrakopoulos et al., 2002; Menabde et al., 2007; Albor and Dimitrakopoulos, 2010; Dimitrakopoulos, 2011; Asad and Dimitrakopoulos, 2013; Marcotte and Caron, 2013; Fricke et al., 2014; Lamghari and Dimitrakopoulos, 2016b).

Over the past years, several exact and approximate methods have been proposed for optimizing mineral value chains. Most of the literature deals with the simplest case described above and referred to as MPS, but recent years have seen the development of solution methods for more complex mineral value chains including multiple mines, multiple processing streams, and accounting for metal uncertainty. A branch and cut algorithm has been developed by Caccetta and Hill (2003) for the deterministic MPS. Bienstock and Zuckerberg (2010) introduced an efficient algorithm to solve the linear relaxation of the problem. Other exact methods that exploit the structure of the problem were proposed by Boland et al. (2009) and Bley et al. (2010). Heuristic and metaheuristic approaches were introduced, among others, by Ferland et al. (2007) and Cullenbine et al. (2011), while hybrid methods were proposed by Moreno et al. (2010), Chicoisne et al. (2012), and Lamghari et al. (2015). The stochasic version of MPS that accounts for metal uncertainty (SMPS) has been tackled mainly using metaheuristics. Godoy (2002) and Albor and Dimitrakopoulos (2009) proposed simulated annealing algorithms. A tabu search algorithm and a variable neighborhood descent algorithm have been developed by Lamghari and Dimitrakopoulos (2012) and Lamghari et al. (2014), respectively.

[1] In practical settings, a medium-size open-pit mine consists of tens thousands of blocks, and the resulting optimization problem has hundreds of thousands binary variables. This is far more than any of the existing exact methods can handle in a reasonable amount of time.

More complex stochastic mineral value chains involving multiple destinations for the extracted material have been studied by Ramazan and Dimitrakopoulos (2013), Goodfellow and Dimitrakopoulos (2014), Behrang et al. (2014), Lamghari and Dimitrakopoulos (2016a), and Montiel and Dimitrakopoulos (2015). Mineral value chains incorporating both open-pit and underground operations have been considered by Montiel et al. (2016). Hoerger et al. (1999); Chanda (2007); Whittle (2007, 2009); and Kawahata et al. (2015) also considered mineral value chains consisting of multiple mines, but their studies are restricted to deterministic environments; i.e., they do not account for metal uncertainty. For a general overview of mine planning optimization problems, see Dimitrakopoulos (2011).

To the best of our knowledge, all the approaches proposed in the literature to optimize mineral value chains either use aggregation techniques to reduce the size of the problem so that the resulting model is of tractable size and can be solved using exact methods, or are based on (meta)heuristics, or combine (meta)heuristics and exact methods. Each of these approaches presents some weaknesses. Aggregation can severely compromise the validity and usefulness of the solution (Bienstock and Zuckerberg, 2010). It causes loss of profitability and might even lead to infeasible solutions (Boland et al., 2009). Approaches based on (meta)heuristics have been proven to be successful for solving large-scale instances without resorting to aggregation, but they have two major flaws. First, they might involve a relatively large number of parameters and/or algorithm choices, and they generally do not provide guidance on how to make such choices. Therefore, it is not always clear a priori which choices will perform better in a particular situation, meaning that tuning might be required if dealing with new instances of the same problem. Second, they are problem-specific methods. Problem-specific methods can often obtain excellent results for the problem they have been designed for, but they are not readily applicable to other problems or other variants of the same problem. They have to be adapted to the new problem, and if so, they might not perform as well as on the original problem. An illustration of such a situation can be found in the study in Lamghari and Dimitrakopoulos (2016a), which indicates that the tabu search metaheuristic developed in Lamghari and Dimitrakopoulos (2012) to solve the SMPS with one processing stream worked well on that particular problem but exhibits a poorer performance on the variant considering multiple destinations for the mined material, including stockpiles (SMPS+). Some approaches that combine (meta)heuristics and exact methods are also limited by the same weakness; that is, they are tailored to one specific case. For example, the algorithm proposed by Moreno et al. (2010) is only applicable to the variant of the MPS with a single resource constraint per period and for which such a constraint is an upper bound.

As is evident from the above discussion, there is a need for solution approaches that are able to tackle large-scale instances without resorting to aggregation, that are self-managed, and that are more general than currently existing methodologies. The latter feature is particularly important since, as mentioned earlier, the components of a mineral value chain vary from one mining operation to another. A general algorithmic framework that is re-usable without major structural modifications is thus more appropriate than a problem-specific approach that must be re-adapted (if this is possible) to each new mineral value chain tackled. In response to this need, this paper proposes the use of hyper-heuristic approaches. Operating at a level of abstraction above that of a metaheuristic, a hyper-heuristic is an emergent search methodology that seeks to automate the process of selecting and combining simpler heuristics or of generating new heuristics from components of existing heuristics in order to solve hard computational search problems (Burke et al., 2003a; Ross, 2005; Burke et al., 2013). It can be seen as an algorithm that tries to find an appropriate solution method at a given decision point rather than a solution. The ideas behind hyper-heuristics date back to the 1960's (Fisher and Thompson, 1963), but the term hyper-heuristic was used only in the late 1990's. It was first used in the context of automated theorem proving to describe a protocol that combines different Artificial Intelligence methods (Denzinger et al., 1997), then independently in the context of combinatorial optimization to describe a high-level heuristic to choose lower-level heuristics using only limited problem-specific information (Cowling et al., 2001). Over the last decade, many papers presenting successful applications of hyper-heuristics to various difficult combinatorial problems have appeared in the literature, including hyper-heuristics for personal scheduling, sports scheduling, educational timetabling, space allocation, cutting and packing, and vehicle routing problems. For a recent and comprehensive survey of the literature, see Burke et al. (2013).

There has been no work investigating hyper-heuristics to solve mine planning optimization problems, and the objective of this paper is to propose such a work. There are two main categories of hyper-heuristics: heuristic selection, which are methodologies for choosing existing heuristics, and heuristic generation which are methodologies for creating new heuristics from a set of components of other existing heuristics (Burke et al., 2010). The three hyper-heuristic approaches proposed in this paper fall under the first category. More specifically, they use a set of simple perturbative low-level heuristics to improve a candidate solution. The decision of which low-level heuristic should be applied at a given step of the search process relies on a score-based learning mechanism; that is, a score is associated with each heuristic reflecting its past performance, and the heuristics are selected based on these scores. The scores

are updated periodically, and the process terminates when a pre-specified stopping criterion is met. While this general framework is similar in the three hyper-heuristic approaches proposed in this paper, the score update rules and the heuristic selection strategy are different. Two approaches are refined versions of ones previously proposed in the literature (Burke et al., 2003b; Drake et al., 2012), while the third one is a novel approach that uses some of the ideas of the first two but also includes new features aimed to overcome their weaknesses. The generality of the three proposed hyper-heuristics is demonstrated by applying them to various instances of two types of mineral value chains having different constraints and problem characteristics. Their performance is assessed by comparing them to each other and to other search methodologies from the literature. This comparison indicates that the third hyper-heuristic outperforms the two others, providing results that are comparable to or improve on the results obtained by the state-of-the-art problem-specific methods.

In the next section, a description of the two mineral value chains considered in this paper is given. The proposed hyper-heuristics are described in Section 3. Numerical results are reported in Section 4. Section 5 provides conclusions and directions for future research.

2 Mineral value chains description

As stated in Section 1, there are different types of mineral value chains depending on their components. The two that are considered in this paper are described in the following sections.

2.1 Mineral value chain with one processor ($SMPS$)

This mineral value chain consists of one open-pit mine from which blocks are extracted,[2] one processing facility where extracted ore blocks are treated to recover the metal they contain,[3] and one waste dump where extracted blocks that cannot be processed profitably are disposed.[4] Although not profitable, low-grade blocks have to be extracted to either have access to higher-grade blocks or ensure safe wall slopes for the pit. The metal content of any given block, which determines whether the block is an ore block to be processed or a waste block to be discarded, is not known prior to the extraction. What is available is a number of equiprobable scenarios, each of which provides possible values of the blocks' metal content given the geological data and the information obtained from drilling. To generate the scenarios, geostatistical techniques of conditional simulation are used. Those can be seen as complex Monte Carlo simulation frameworks able to reproduce all available data and information, as well as spatial statistics of the data (Goovaerts, 1997; Chiles and Delfiner, 2012; Rossi and Deutsch, 2014; Maleki and Emery, 2015; Horta and Soares, 2010; Boucher and Dimitrakopoulos, 2009).

The optimization problem associated with the mineral value chain described above, referred to as SMPS in the rest of the paper, concerns the design of a mining sequence over a discrete finite planning horizon (the life-of-the-mine); that is, deciding which blocks should be extracted at each period of the life-of-the-mine. In doing so, various constraints must be satisfied. Logical and physical restrictions impose that each block can be extracted at most once, after all its predecessors have been extracted. Operational restrictions require that, at each period of the life-of-the-mine, the total amount of material extracted (ore and waste), the total amount of ore processed, and the total amount of metal produced should lie between specific lower and upper bounds. The extraction and processing operations incur costs, while the metal recovered from processing is sold and generates revenue. All cost and revenue components are future cash-flows and thus must be discounted to the present, so feasible solutions are typically evaluated by their net present value (NPV) to select the one that provides the highest value. As mentioned earlier, extraction decisions are to be made prior to knowing the metal content of the blocks, and the latter affects: i) the amount of ore available for processing at each period; ii) the amount of metal produced from processing at each period; and iii) the NPV. Thus, according to the particular metal scenario realized, not only might the NPV shift upward or downward, but also ore and metal production targets might fail to be satisfied in some or all periods (exceed the upper bound or fall under the lower bound). Some recourse actions are available to adapt to the situation at hand, but they are subject to extra costs. For example, if an excess in ore production occurs, extra processing capacity is required and an additional cost is incurred.

[2] Recall that the mine is discretized into a set of blocks, each of which represents a volume of material that can be mined.

[3] This process incurs a unit cost, so a block is processed only if the revenue from the metal recovered pays for the processing and selling costs. Such a block is referred to as an ore block.

[4] Low-grade blocks, also referred to as waste blocks.

Clearly, for better-informed decision-making, the aforementioned effects of uncertainty must be taken into account. Because initially one has to decide on which blocks to extract, but only later, when the metal uncertainty is disclosed, does one have to decide how best to deal with the excess and shortage in ore and metal (recourse decisions), the problem is formulated as a two-stage stochastic programming model (Birge and Louveaux, 2011), where the overall objective is to maximize the expected net present value of the mining operation and to minimize the future expected recourse costs over the uncertain metal scenarios. This model is described in detail in Lamghari and Dimitrakopoulos (2012) and is briefly recalled in Appendix A.

2.2 Mineral value chain with multiple processors and stockpiles ($SMPS+$)

Unlike the mineral value chain described in the previous section that consists of one processor, this mineral value chain consists of multiple processors and stockpiles. The stockpiles are used to absorb the excess of ore such that when such a situation occurs, some ore is not immediately processed in the period it is mined in but rather sent to the stockpiles from which it is reclaimed in periods where there is spare capacity. Hence, in each period, an extracted block is sent either to one of the processors, or to one of the stockpiles, or to the waste dump. If blocks are sent to the stockpile, unit transportation and handling costs are incurred. Costs are also incurred when reclaiming material from the stockpiles. Thus, an optimal solution maximizes the NPV and indicates the set of blocks that should be extracted in each period, the destination of these blocks, and the amount of material to take from the stockpiles in each period to feed the processors. That is, compared to the optimization problem described in the previous section, this problem incorporates the material flow aspect in addition to the mining sequence design.

In the rest of the paper, we will refer to the optimization problem associated with this particular mineral value chain as SMPS+. SMPS+ can also be formulated as a two-stage stochastic program. The first-stage consists of designing the mining sequence, and the second-stage consists of processing and stockpiling (i.e.; material flow decisions). The second stage decisions are made based on the first-stage decisions (i.e, the mining sequence) and on the realized metal scenario. A detailed description of the mathematical model is available in Lamghari and Dimitrakopoulos (2016a), and an outline of it is provided in Appendix B.

3 Hyper-heuristic solution approaches

As mentioned in Section 1, the three hyper-heuristic solution approaches proposed in this paper fall under the category of *heuristic selection* (Burke et al., 2010); that is, they are methodologies for choosing existing heuristics. Burke et al. (2010) further categorize hyper-heuristics according to the nature of the heuristic search space (constructive heuristics versus perturbation heuristics) and the source of feedback during learning (online learning, offline learning, and no learning). With respect to this classification, the three proposed approaches can be seen as approaches based on perturbation low-level heuristics with online learning.

The general framework can be summarized as follows: The algorithm starts by generating an initial solution (a random feasible solution in this paper), and then tries to iteratively improve it using different local search heuristics (low-level heuristics). These heuristics are described in Section 3.3. To determine the appropriate heuristic to apply at a given iteration, the algorithm relies on a score-based learning mechanism. This means that a periodically updated score $S(h_j)$ is assigned to each heuristic h_j to measure how well h_j has performed during the search, and the heuristics are selected based on these scores. Different strategies for the heuristic selection process can be used, and each leads to a different hyper-heuristic. The strategies proposed in this paper are described in detail in Sections 3.1 and 3.2. The selected heuristic is applied to the current solution once to obtain a new solution. Another decision that has to be made at this point is whether or not to accept this new solution. In this paper, the new solution is always accepted, independently of its quality. This means that this new solution becomes the new current solution. It also replaces the incumbent solution if it has a better objective value. This procedure is iterated until the stopping criterion is met. In this paper, the procedure terminates when a specified number of iterations, Υ_{max}, has elapsed.

3.1 Refined versions of hyper-heuristics proposed in the literature

This section describes two previously developed hyper-heuristics that have been refined to improve their performance.

3.1.1 Tabu search hyper-heuristic

The first hyper-heuristic, henceforth referred to as HH1, was proposed by Burke et al. (2003b). To guide the heuristic selection process, HH1 uses principles of reinforcement learning and tabu search metaheuristic. Let H be the number of low-level heuristics. Initially, each heuristic has a score equal to 0. As the search progresses, the scores increase and decrease within the interval $[0, H]$, and the heuristics are selected according to the updated scores. Not all heuristics are available for selection at a given iteration. A dynamic tabu list of heuristics is maintained to temporarily exclude some of them. Details of a typical iteration are as follows: The hyper-heuristic selects the non tabu low-level heuristic having the highest score. It applies it once and then compares the value of the current solution to the value of the new solution. If the new solution is better than the current solution, the heuristic is rewarded by incrementing its score by one. If the new solution and the current solution have the same value, the heuristic is punished by decrementing its score by one and making it tabu. Finally, if the new solution is worse than the current solution, the hyper-heuristic proceeds as in the previous case except that the tabu list is first emptied before adding the heuristic.

Two possible implementations of the hyper-heuristic HH1 described above were considered. In one the same strategy proposed in Burke et al. (2003b) to update the tabu list was used; that is, heuristics are included in the tabu list on a first-in-first-out basis, and the tabu list is emptied whenever a solution worse than the current one is obtained. The authors justify emptying the tabu list by claiming that there is no point in keeping a heuristic tabu once the current solution has been modified. However, there is a potential drawback to this strategy: The tabu status is revoked too soon, which might lead to choosing a relatively poor heuristic too often, thereby missing the opportunity to apply other better performing heuristics. To clarify, since the heuristics are rewarded the same way, independently of the magnitude of improvement they can achieve, a heuristic that is able to slightly improve the solution but also deteriorates it occasionally can gain large rewards. It might have the highest score and making it non tabu as soon as one single other heuristic has also been found deteriorating leads to using the same heuristic again and again (cycling behavior). This implies that the other heuristics, which might be better performing and more appropriate at the current decision point, have little or no chance to be selected. In the second implementation of HH1, this drawback is overcome as follows: Whenever a heuristic is not able to improve the current solution, it is made tabu. The tabu tenure (γ) is chosen randomly within a specified interval $[\Gamma_{min}, \Gamma_{max}]$, and the tabu list is emptied only if all heuristics are tabu. Moreover, for the first H iterations of the algorithm (H being the number of low-level heuristics), heuristics are not selected based on their scores and their tabu status but rather randomly, making sure that each heuristic is selected only once. This refined version of HH1 was found to obtain better results than the original version. We thus report in Section 4 only results obtained with it. Algorithm 1 shows the pseudocode of the refined version of HH1.

Algorithm 1 Refined version of HH1

```
Initialization
  Generate an initial solution x
  Set x* := x
  for each heuristic h_j do
      Set S(h_j) := 0
  end for
Improvement
  Stage I: Giving a chance to all heuristics
  Add all heuristics to a list (list of not yet selected heuristics)
  while the number of heuristics in the list > 0 do
      Choose a heuristic h_j randomly from the list
      Generate a new solution x' from x using h_j
      if x' is better than x* then
          Set x* := x'
      end if
      if x' is better than x then
          Set S(h_j) := S(h_j) + 1
      end if
      Set x := x'
      Remove h_j from the list
  end while
```

Stage II: Selecting heuristics based on their score and tabu status
while stopping criterion not met **do**
  if all heuristics are tabu **then**
    Revoke the tabu status of all heuristics
  end if
  Choose a heuristic h_j that maximizes $S(h_j)$ and is not tabu (ties are broken up randomly)
  Generate a new solution x' from x using h_j
  if x' is better than x^* **then**
    Set $x^* := x'$
  end if
  if x' is better than x **then**
    Set $S(h_j) := \max(S(h_j) + 1, H)$ (H being the number of low-level heuristics)
  else
    Set $S(h_j) := \min(S(h_j) - 1, 0)$
    Generate a random number γ in $[\Gamma_{min}, \Gamma_{max}]$
    Make h_j tabu for γ iterations
  end if
  Set $x := x'$
end while

return x^*.

3.1.2 Choice-function hyper-heuristic

The second hyper-heuristic considered in this paper, henceforth referred to as HH2, has been proposed by Drake et al. (2012). It extends the hyper-heuristic developed in Cowling et al. (2001) and differs from HH1 in that i) it does not incorporate any mechanism at the high level to prevent choosing heuristics that did not perform well recently; and ii) it uses a more complex score update scheme. To be more specific, rather than incrementing and decrementing the score based on the heuristic's ability to improve the solution as HH1 does, HH2 calculates the scores as a weighted sum of the three following measures:

- The first measure, f_1, keeps track of the performance of the heuristics. It accounts for the improvement that each heuristic has achieved so far as well as the time it has required. Following the same notation as in Drake et al. (2012), let $I_n(h_j)$ be the change in the objective function value obtained the n^{th} last time h_j was called (applied), and let $T_n(h_j)$ be the time required. Denote by $\phi \in]0,1[$ a weight adjustment parameter, defining the importance given to recent performance. The value of $f_1(h_j)$ is computed using the following formula:

$$f_1(h_j) = \sum_n \phi^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \tag{1}$$

- The second measure, f_2, seeks to capture any pairwise dependencies between heuristics. Whenever h_j is called immediately after h_k, the value of $f_2(h_k, h_j)$ is updated as follows:

$$f_2(h_k, h_j) = \sum_n \phi^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)} \tag{2}$$

  where, ϕ is as defined previously, and $I_n(h_k, h_j)$ and $T_n(h_k, h_j)$ are respectively the change in the objective function value and the time required by heuristic h_j at the n^{th} last call following a call to h_k.
- The last measure, f_3, accounts for the time elapsed since each heuristic was last selected. If we denote this time by $\tau(h_j)$, then:

$$f_3(h_j) = \tau(h_j) \tag{3}$$

The score associated with heuristic h_j is a weighted sum of the three measures (h_k being the heuristic called immediately before h_j):

$$S(h_j) = \phi f_1(h_j) + \phi f_2(h_k, h_j) + \delta f_3(h_j). \tag{4}$$

Note that the purpose of using measures f_1 and f_2 is to intensify the search by favoring heuristics that have shown good performance, while measure f_3 aims to give all heuristics a chance to be selected, thus providing an element of diversification. The weights ϕ and δ are parameters in the interval]0,1[used to provide a balance between intensification and diversification. They are dynamically adjusted during the search process based on reinforcement learning principles. This is done as follows: Once the selected heuristic has been applied, the new solution is compared to the current solution. If it is better, then ϕ is rewarded by increasing its value to ϕ_{max}, a maximum value close to the upper bound 1, while δ is decreased to δ_{min}, a minimum value close to the lower bound 0, thus promoting intensification but reducing diversification. Otherwise, the value of ϕ is decreased by a linear factor κ and δ is increased by the same factor to gradually favor diversification over intensification. In Drake et al. (2012), the authors used the values 0.99, 0.01, and 0.01 for the parameters ϕ_{max}, δ_{min}, and κ, respectively. In the numerical results presented in Section 4, we used the values 0.9, 0.1, and 0.1, as preliminary tests showed that, for the problems addressed in this paper, these values provide better results than the values used in Drake et al. (2012).

A particularity of the problems addressed in this paper is that their objective functions take very large values in the order of hundreds of millions. Preliminary tests showed that, in general, the change in the objective function resulting from applying a given heuristic is in the order of tens of thousands, and that this change is obtained in a fraction of a second. Consequently, in Equation (4), the values of the intensification components f_1 and f_2 have an order of magnitude much larger than that of the diversification component, f_3. The latter component is dominated by the first two and thus becomes obsolete when calculating the scores. To overcome this weakness, we made the two following refinements. First, as in the refined version of HH1, in a first stage of the algorithm, a chance is given to all heuristics to improve the solution; that is, the heuristics are selected randomly, making sure that each heuristic is selected only once, and the values of f_1, f_2, and f_3 are updated accordingly. Afterwards, heuristic selection is performed based on the scores, which are computed using Equation (4), but f_1, f_2, and f_3 are first normalised to a value in the interval [1,10] before using them in Equation (4). This version of the hyper-heuristic was found to obtain better results than the original version, as it provides a better balance between intensification and diversification. We thus report in Section 4 only results obtained with this version. An outline of the pseudocode of the refined version of HH2 is given in Algorithm 2.

Algorithm 2 Refined version of HH2

Initialization

Generate an initial solution x
Set $x^* := x$
Set $\phi := 0.5$, $\delta := 0.5$, $\phi_{max} := 0.9$, $\delta_{max} := 0.9$, $\phi_{min} := 0.1$, $\delta_{min} := 0.1$, and $\kappa := 0.1$

Improvement

Stage I: Giving a chance to all heuristics

Add all heuristics to a list (list of not yet selected heuristics)
while the number of heuristics in the list > 0 **do**
    Choose a heuristic h_j randomly from the list
    Generate a new solution x' from x using h_j
    if x' is better than x^* **then**
        Set $x^* := x'$
    end if
    Calculate $f_1(h_j)$ and $f_2(h_k, h_j)$ using equations (1) and (2), respectively
    Set $f_3(h_j) := 0$ and update the f_3 values corresponding to the remaining heuristics
    Set $x := x'$
    Set $k := j$
    Remove h_j from the list
end while

Stage II: Selecting heuristics based on their score
while stopping criterion not met **do**
Normalize f_1, f_2, and f_3
for each heuristic h_j **do**
Calculate $S(h_j)$, the score of h_j, using equation (4)
end for
Choose a heuristic h_j that maximizes $S(h_j)$ (ties are broken up randomly)
Generate a new solution x' from x using h_j
if x' is better than x^* **then**
Set $x^* := x'$
end if
if x' is better than x **then**
Set $\phi := \phi_{max}$, $\delta := \delta_{min}$
else
Set $\phi := \min\{\phi - \kappa, \phi_{min}\}$, $\delta := \max\{\delta + \kappa, \delta_{max}\}$
end if
Update $f_1(h_j)$ and $f_2(h_k, h_j)$ according to equations (1) and (2), respectively
Set $f_3(h_j) := 0$ and update the f_3 values corresponding to the remaining heuristics
Set $x := x'$
Set $k := j$
end while

return x^*.

Even though it is better than the original version, the refined version of HH2 still has two drawbacks:

- It might select a heuristic that largely deteriorates the solution and requires long computational time rather than a heuristic that slightly deteriorates the solution and requires short computational time. To clarify, consider the following scenario with only two low-level heuristics, h_1 and h_2. Recall that the heuristics are selected randomly at the first stage of the algorithm. Assume that h_1 was first applied and resulted in $I_1(h_1) = -20$ and $T_1(h_1) = 10$. Afterwards, h_2 was selected and resulted in $I_1(h_2) = -1$ and $T_1(h_2) = 1$. Now, at the second stage of the algorithm, the heuristics are selected based on their scores. The scores of h_1 and h_2, after normalising f_1, f_2, and f_3, are $S(h_1) = 11\,\phi + 10\delta$ and $S(h_2) = 11\,\phi + \delta$, respectively. Clearly, $S(h_1) > S(h_2)\ \forall \phi, \delta \in]0,1[$. So, despite the fact that h_2 showed a relatively better performance compared to h_1, the latter has the highest score and will be selected.
- Although measures f_1 and f_2 are defined so as to give a greater importance to recent performance, the way the parameter ϕ is adjusted can lead to early performance dominating recent performance. Consider two heuristics h_j and h_k. Assume that h_j obtained large improvements in the early stages of the search but has exhibited a poor performance recently, while h_k has yielded small improvements since the beginning of the search. Assume also that the last iteration resulted in an improvement of the current solution and, consequently, the value of ϕ was increased to ϕ_{max} to favor heuristics that showed good performance and intensify the search. Now recall that the parameter ϕ defines not only the importance given to good performing heuristics but also the importance given to previous performance; i.e, large ϕ values put more emphasis on previous performance. So, looking back to the example above, while one would want to favor h_k because the improvements it achieves, although small, are more significant at the current stage of the search, HH2 will select h_j because its recent performance is dominated by its early performance and thus has a small impact on the score value.

The two aforementioned issues are addressed in the new proposed hyper-heuristic presented in the next section.

3.2 New proposed hyper-heuristic

This hyper-heuristic, referred to as HH3 in the rest of the paper, uses some of the ideas of HH1 and HH2 but also includes new features aimed to overcome their weaknesses, outlined in the previous sections.

Similar to the refined versions of HH1 and HH2, HH3 proceeds in two stages. In the first stage, the algorithm randomly picks a heuristic h_j, applies it, returns the resulting change in the objective function value ($\Delta f(h_j)$) as well

as the time required ($T(h_j)$), and computes the heuristic's initial score ($S(h_j)$). The initial scores reflect the order of importance given to the heuristics. Of first importance are heuristics that are able to improve the solution. The larger the improvement rate per unit of time is, the more important the heuristic is considered. Because heuristics that deteriorate the solution help getting out of local optima, they are considered more important than heuristics that cannot modify the objective function value. However, not all of them are equally important. The more a heuristic deteriorates the solution and the more computational time it requires, the less important it is considered. Accordingly, the initial scores are computed by the following formula:

$$S(h_j) = \begin{cases} \dfrac{\Delta f(h_j)}{T(h_j)} & \text{if } \Delta f(h_j) \geq 0, \\ \dfrac{1}{|\Delta f(h_j)| T(h_j)} & \text{otherwise.} \end{cases} \tag{5}$$

The first stage of the algorithm terminates once all low-level heuristics have been considered. In the second stage, HH3 selects the heuristics based on two factors: the heuristics' scores and their tabu status. While the tabus are managed and used in the same way they are in the refined version of HH1 (see Section 3.1.1), the strategy for heuristic selection is different. Rather than selecting the non tabu heuristic having the highest score as HH1 does, HH3 uses a roulette-wheel strategy. It associates with each non tabu heuristic h_j a selection probability p_j calculated by dividing its score by the total score of the non tabu heuristics ($p_j = \frac{S(h_j)}{\sum_{k:h_k non\ tabu} S(h_k)}$). It then randomly selects a heuristic based on these probabilities. Another noticeable difference between HH3 and the two hyper-heuristics described in the previous sections is the frequency at which the scores are updated and the score update scheme. In HH3, the scores are updated every ζ iterations, accounting for the average performance of the heuristics during these iterations. For this purpose, the algorithm maintains for each heuristic h_j two measures, $\pi_1(h_j)$ and $\pi_2(h_j)$. Such measures are initially equal to 0. Whenever h_j is applied, either $\pi_1(h_j)$ or $\pi_2(h_j)$ is increased. The increase is related to the obtained change in the objective function value ($\Delta f(h_j)$). Specifically, if $\Delta f(h_j) > 0$ (i.e., if h_j improves the current solution), $\frac{\Delta f(h_j)}{T(h_j)}$is added to $\pi_1(h_j)$; if $\Delta f(h_j) < 0$ (i.e., if h_j deteriorates the current solution), $\frac{1}{|\Delta f(h_j)| T(h_j)}$ is added to $\pi_2(h_j)$; and if $\Delta f(h_j) = 0$ (i.e., if h_j cannot modify the value of the current solution), both $\pi_1(h_j)$ and $\pi_2(h_j)$ remain unchanged. Let $\eta(h_j)$ be the number of times heuristic h_j has been selected in the last ζ iterations, and α and β be two weight adjustment parameters in $[0, 1]$. The scores are recalculated as follows:

$$S(h_j) := \begin{cases} S(h_j) & \text{if } \eta(h_j) = 0, \\ (1-\alpha) S(h_j) + \alpha \dfrac{\beta \pi_1(h_j) + (1-\beta)\pi_2(h_j)}{\eta(h_j)} & \text{otherwise.} \end{cases} \tag{6}$$

Clearly, α defines the importance given to recent performance, while β defines the importance given to heuristics that were recently able to improve the solution. In this paper, the value of α is set to 0.7 to decrease the weight of previous performance (recall that one of the weaknesses of HH2 is that early performance sometimes dominates recent performance). On the other hand, the parameter β is self-adjusted during the search process. The value of β is initially set equal to 0.5, and it is modified every ζ iterations based on whether or not a new incumbent solution has been found during the last segment of search: If a new solution better than the incumbent is found during the last ζ iterations, β is increased to 1; otherwise, it is decreased to $\max(\beta - 0.1, 0)$. This way of proceeding ensures that emphasis is put on intensification if a new incumbent is found, while focus is gradually shifted to diversification otherwise. Indeed, when the value of β is increased, the score of heuristics that were recently able to improve the solution is also increased, so such heuristics are more likely to be selected, leading to an intensification of the search. As the value of β decreases, the effect is the opposite, leading to a diversification of the search. Once the value of β is updated, the scores are calculated with Equation (6), $\pi_1(h_j), \pi_2(h_j)$ and $\eta(h_j)$ are reset to zero for each h_j, the tabu list is emptied, and a new segment of search is initiated for another ζ iterations. This process is repeated until the stopping criterion is met. An overview of the solution procedure is provided in Algorithm 3.

Algorithm 3 HH3

Initialization

Generate an initial solution x
Set $x^* := x$
Set $\alpha := 0.7$, $\beta := 0.5$, and $newIncumbent :=$ **false**

Improvement

Stage I: Giving a chance to all heuristics

Add all heuristics to a list (list of not yet selected heuristics)
while the number of heuristics in the list > 0 **do**
    Choose a heuristic h_j randomly from the list
    Generate a new solution x' from x using h_j
    if x' is better than x^* **then**
        Set $x^* := x'$
        Set $newIncumbent :=$ **true**
    end if
    Calculate the initial score of h_j using equation (5)
    Set $x := x'$
    Remove h_j from the list
end while

Stage II: Selecting heuristics based on their score and tabu status

$iter := 1$
for each heuristic h_j **do**
    Set $\pi_1(h_j) := 0$, $\pi_2(h_j) := 0$, and $\eta(h_j) := 0$
end for
while stopping criterion not met **do**
    if all heuristics are tabu **then**
        Revoke the tabu status of all heuristics
    end if
    Choose, among the heuristics that are not tabu, a heuristic h_j using roulette-wheel selection based on scores
    Set $\eta(h_j) := \eta(h_j) + 1$
    Generate a new solution x' from x using h_j
    if x' is better than x^* **then**
        Set $x^* := x'$
        Set $newIncumbent :=$ **true**
    end if
    if x' is better than x **then**
        Update $\pi_1(h_j)$
    else
        Update $\pi_2(h_j)$
        Generate a random number γ in $[\Gamma_{min}, \Gamma_{max}]$
        Make h_j tabu for γ iterations
    end if
    if $iter < \zeta$ **then**
        Set $iter := iter + 1$
    else
        if $newIncumbent =$ **true then**
            Set $\beta := 1$
        else
            Set $\beta := \max(\beta - 0.1, 0)$
        end if
        Update the score of all heuristics using equation (6)
        Revoke the tabu status of all heuristics
        for each heuristic h_j **do**
            Set $\pi_1(h_j) := 0$, $\pi_2(h_j) := 0$, and $\eta(h_j) := 0$
        end for
        Set $iter := 1$
        Set $newIncumbent :=$ **false**
    end if
    Set $x := x'$
end while

return x^*.

3.3 Low-level heuristics

To produce new solutions, the three hyper-heuristics described in the previous section use 24 simple perturbative low-level heuristics, each of which examines a subset of one of the following four neighborhoods, previously proposed in the literature:

- *Single-Shift* (Lamghari and Dimitrakopoulos, 2012): This neighborhood involves moving a single block from its current period t to another period $t' \neq t$.
- *Swap* (Lamghari et al., 2014): This neighborhood allows exchanging blocks between periods and can be seen as two simultaneous changes associated with the Single-Shift neighborhood. More specifically, it involves moving two blocks: block i from its current period t to another period $t' \neq t$ and another block i' from t' to t.
- *Shift-Before* (Lamghari et al., 2014): Here multiple blocks; namely, a block i and its predecessors mined in the same period, are moved from their current period $t \neq 1$ to the preceding period $(t - 1)$. Recall that a predecessor of block i is a block that has to be extracted to have access to i. In what follows, we will refer to the set formed by a block i and its predecessors mined in the same period as the *inverted cone* whose base is i.
- *Shift-After* (Lamghari et al., 2014): This neighborhood also allows moving multiple blocks. A block and its successors mined in the same period are moved from their current period t to the next period $(t + 1)$. Note that j is a successor of i if and only if i is a predecessor of j. In what follows, we will refer to the set formed by a block i and its successors mined in the same period as the *cone* whose apex is i.

Not only do the proposed heuristics examine different subsets of the four neighborhoods described above (different sub-neighborhoods), but they also use different functions to evaluate solutions in these sub-neighborhoods and different strategies to select one of them to become the new current solution. Generating only subsets of the neighborhoods and using different evaluation functions and selection strategies serves three main purposes: to reduce the computational effort, to drive the search to interesting parts of the search space, and to ensure intensification and diversification.

To simplify the discussion, the heuristics are classified in three different groups. The first group contains heuristics that select block(s) from a random period and move them either earlier or later. The second group contains heuristics that select block(s) from a specific period, as opposed to a random period, and move them either earlier or later. Clearly, heuristics in these two groups do not allow for any changes in the set of extracted blocks. Heuristics in the third group allow for such changes by either dropping block(s) from the schedule or adding unscheduled block(s) to the schedule. Heuristics in the three groups consider only moves that yield a feasible solution. Below, additional details about the heuristics are provided.

Heuristics that choose blocks from a random period

- h_1: This heuristic explores a subset of the *Single-Shift* neighborhood. It starts by randomly selecting a period t. It then identifies blocks currently scheduled in *t* that can be moved either earlier or later without violating the constraints. Moves are evaluated based on the change produced in the objective function value, and the best move is selected.
- h_2: Similar to h_1 except that it considers only blocks that can be moved earlier. Furthermore, the evaluation of a move is based on the total economic value of the block and all its successors. This evaluation function can be seen as a measure of attractiveness used to identify potential blocks that if advanced will entail advancing high-grade ore blocks. Thus, h_2 explores a smaller subset of the Single-Shift neighborhood compared to h_1, and to orient the search, it does not use the objective function of the problem but an auxiliary function, the value of the block and all its successors.
- h_3: Unlike the two previous heuristics, which move a single block, this heuristic simultaneously moves multiple blocks; more specifically, it advances the extraction of an inverted cone whose base block is currently scheduled in t, from t to $t - 1$. Only inverted cones having a positive economic value are considered, and the heuristic selects the one with the highest unit economic value. Thus, h_3 explores a subset of the *Shift-Before* neighborhood, and to orient the search, it uses another auxiliary function, the unit economic value.

- h_4: Similar to h_3 except that all inverted cones are considered, among which one is chosen at random. This heuristic induces some form of diversification.
- h_5: Similar to h_4, but the moves are evaluated based on the change produced in the objective function value, and the best move is selected.
- h_6: This heuristic also allows simultaneously moving multiple blocks. However, rather than changing the period of an inverted cone from t to $t - 1$, it changes the period of a cone whose apex is currently scheduled at t, from t to $t + 1$, which means that it explores a subset of the *Shift-After* neighborhood. Only cones having a non-positive economic value are considered, and the heuristic selects the one with the smallest unit economic value.
- h_7: Similar to h_6 except that all cones are considered, among which one is chosen at random. Like h_4, h_7 is used for diversification purposes.
- h_8: Similar to h_7, but the moves are evaluated based on the change produced in the objective function value, and the best move is selected.
- h_9: Similar to h_8, but the blocks to be moved are not necessarily related to each other via precedence, and they are moved sequentially. At each iteration, a single block is selected and moved from t to $t + 1$, and this process is repeated as long as there is improvement in the objective function value. Therefore, h_8 explores a subset of the *Single-Shift* neighborhood using a best-improvement descent. It acts as a trimming mechanism to free some capacity in t for hopefully more interesting blocks.
- h_{10}: This heuristic also moves blocks that are not related to each other via precedence. It exchanges blocks i and i' currently scheduled in periods t and $t + 1$, respectively. That means that h_{10} explores a subset of the *Swap* neighborhood. Moves are evaluated based on the change produced in the objective function value and selected using a first improving strategy.

Heuristics that choose blocks from a specific period

Unlike the previous heuristics ($h_1 - h_{10}$), the following three heuristics do not select blocks from a random period but rather from a specific period in an attempt to reduce either soft constraints violations or the tightness of the hard constraints. The first heuristic explores a subset of the *Single-Shift* neighborhood, while the last two explore subsets of the *Shift-After* or *Shift-Before* neighborhoods. The way these subsets are chosen and explored is explained below.

- h_{11}: This heuristic first identifies the period with the highest penalty cost (incurred by violation of the soft constraints). It moves a single block currently mined in t either later or earlier. The moves are evaluated based on the change produced in the objective function value, and the best move is selected.
- h_{12}: This heuristic first identifies the period with the highest mining utilization, t (the mining utilization is calculated as the total amount mined in period t in the current solution divided by the mining capacity). It then determines the adjacent period with the most residual capacity, t'. If $t' = t - 1$, then the heuristic selects an inverted cone whose base is currently scheduled in t; otherwise (i.e., if $t' = t + 1$), it selects a cone whose apex is currently scheduled in t. The (inverted) cone is selected at random and its period is changed from t to t'.
- h_{13}: Similar to h_{12} except for the way t is selected. Here t is selected among the periods with high penalty cost, not among those with high mining utilization. Also, t is not chosen in a greedy manner but using roulette wheel selection.

Heuristics that modify the set of scheduled blocks

The heuristics described above change the periods of scheduled blocks. However, they do not allow for any changes in the set of extracted blocks. Such changes are obtained with the heuristics presented below.

- h_{14}: This heuristic starts by identifying blocks that are currently unscheduled then adds one of them to the schedule. Moves are evaluated based on the change produced in the objective function value, and the best one is selected.
- h_{15}: Similar to h_{14}, but only improving moves are considered.
- h_{16}: Similar to h_{14} but more aggressive in the sense that it induces greater solution changes than h_{14}. It adds to the schedule inverted cones (a block and its predecessors) rather than a single block.
- h_{17}: Similar to h_{16}, but only improving moves are considered.

- h_{18}: This heuristic considers only inverted cones having a positive economic value, among which it selects the one with the highest unit economic value.

The following heuristics drop block(s) from the schedule.

- h_{19}: His heuristic drops a single block from the schedule. Moves are evaluated based on the change produced in the objective function value. Only improving moves are considered, and the best one is selected.
- h_{20}: Similar to h_{19}, but it does not terminate after dropping a single block. The process is repeated until no improvement is possible.
- h_{21}: Like h_{20}, this heuristic also drops multiple blocks from the schedule. However, these blocks are related to each other by precedence. It removes a cone whose apex is currently scheduled in the last period of the horizon (a block and its successors). Moves are evaluated based on the change produced in the objective function value, and the best one is selected.
- h_{22}: Similar to h_{21} except that only improving moves are considered.
- h_{23}: Similar to h_{21}, but evaluates moves based on the unit economic value of the cones. Only cones having a non-positive value are considered, and the one with the smallest value is selected.
- h_{24}: Similar to h_{19}, but also simultaneously drops a single block from the schedule. In other words, it exchanges an unscheduled block with a scheduled block. The neighborhood is explored using a first-improving strategy.

4 Numerical results

To assess the efficiency and the robustness of the three proposed hyper-heuristic approaches, numerical experiments have been performed on four benchmark test sets, including a total of 33 instances of different sizes and characteristics. These benchmark datasets are briefly described below and summarized in Table 1. The first three are those in Lamghari and Dimitrakopoulos (2016a), while the fourth one is a new dataset that contains larger instances.

Table 1: Overview of the instances in the four benchmark datasets

Dataset	S1	S2	S3	S4
Number of instances	10	3	10	10
Number of blocks (N)	$4{,}273 \leq N \leq 40{,}762$	$14{,}118 \leq N \leq 48{,}821$	$21{,}965 \leq N \leq 22{,}720$	40,900
Number of periods (T)	$3 \leq T \leq 13$	$6 \leq T \leq 16$	$11 \leq T \leq 12$	21
Number of scenarios (S)	20	$20 \leq S \leq 25$	20	20
Number of processors (P)	1	2	2	2
Number of stockpiles	1	2	2	2
Metal type	Copper and Gold	Copper and Gold	Copper	Copper
Block weight (w_i) in tonnes	$5{,}625 \leq w_i \leq 10{,}800$	$5{,}625 \leq w_i \leq 10{,}800$	10,000	10,000
Mining capacity (W^t)	$\left\lceil 1.20 \frac{\sum_{i=1}^{N} w_i}{T} \right\rceil$	$\left\lceil 1.20 \frac{\sum_{i=1}^{N} w_i}{T} \right\rceil$	$\left\lceil \frac{\sum_{i=1}^{N} w_i}{T} \right\rceil$	$\left\lceil \frac{\sum_{i=1}^{N} w_i}{T} \right\rceil$
Processing capacity (Θ_p^t)	$\left\lceil 1.05 \frac{\sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s \theta_{ips} w_i}{T} \right\rceil$	$\left\lceil \frac{\sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s \theta_{ips} w_i}{T} \right\rceil$	$\left\lceil \frac{\sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s \theta_{ips} w_i}{T} \right\rceil$	$\left\lceil \frac{\sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s \theta_{ips} w_i}{T} \right\rceil$

- The first set of benchmark instances, S1, consists of 10 small to large size instances from two real deposits: a copper deposit and a gold deposit. They all contain one processor and one stockpile. Each period is one year long, and it is assumed that the production capacities are identical in all periods. For each instance, it is possible to extract a total of $W^t = \left\lceil 1.20 \frac{\sum_{i=1}^{N} w_i}{T} \right\rceil$ tonnes per year (i.e., $1.20 \frac{\text{total tonnage}}{\text{Number of periods}}$), of which the waste is sent to the waste dump (having an unlimited capacity), and the ore is sent to a processor p (having a capacity of $\Theta_p^t = \left\lceil 1.05 \frac{\sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s \theta_{ips} w_i}{T} \right\rceil$; i. e., $1.05 \frac{\text{Expected amount of ore}}{\text{Number of periods}}$).
- The second set of benchmark instances, S2, consists of three instances representing three different real deposits: two copper deposits and a gold deposit. The size of these instances is larger than those in the first benchmark set. Furthermore, the instances in this set contain two processors and two stockpiles (as opposed to one processor and one stockpile in the first set). Finally, the processing capacities are set to a value 5% smaller than for the instances in the first set so as to make the satisfaction of the processing constraints more difficult and thus force the use of the stockpiles.
- The third set of instances, S3, consists of 10 medium-size instances from a copper deposit with two processors and two stockpiles. They are similar to those in the second set, S2, except for the mining capacities, which are much tighter here. They are set to a value 20% smaller than for the instances in the first and second sets (i.e., $W^t = \left\lceil \frac{\sum_{i=1}^{N} w_i}{T} \right\rceil$).
- The fourth set of instances, S4, consists also of 10 instances from a copper deposit with two processors and two stockpiles. They are similar to those in the third set, S3, except that they are larger.

All algorithms were coded in C++ and the experiments were run on on an Intel(R) Xeon(R) CPU X5675 computer (3.07 GHz) with 96 Go of RAM running under Linux.

4.1 Results for the first set of benchmark instances

In this section, we examine how the three hyper-heuristic approaches (HH1, HH2, and HH3), described in Section 3 and summarized in Algorithms 1-3, perform on the ten bench- mark instances in the set S1. We compare the hyper-heuristics to each other and also to other problem-specific methods that have been recently proposed in the literature. These methods are the tabu search heuristic proposed in Lamghari and Dimitrakopoulos (2012) (TS), the variable neighborhood descent heuristic proposed in Lamghari et al. (2014) (VND), and the network-flow based algorithm presented in Lamghari and Dimitrakopoulos (2016a) (NFA).

Recall that the three hyper-heuristics terminate when a specified number of iterations, Υ_{max}, has elapsed. We set $\Upsilon_{max} = 1000 + 0.25N$, N being the number of blocks. This value was selected based on preliminary tests. Preliminary tests also showed that it is preferable to set $\Gamma_{min} = 0.5H$ and $\Gamma_{max} = H$ (recall that H is the number of low-level heuristics and that $[\Gamma_{min}, \Gamma_{max}]$ is used to choose the tabu tenure; that is, the number of iterations during which a heuristic that has not performed well recently is made tabu). So, these values are used for both HH1 and HH3. For TS and NFA, we used the same parameter settings as in the original papers (Lamghari and Dimitrakopoulos, 2012, 2016a). VND does not have any parameters. All 6 methods (HH1, HH2, HH3, TS, VND, and NFA) start with a random initial solution generated using the heuristic in Lamghari and Dimitrakopoulos (2012), and also make other random choices during the improvement phase. Hence, each of them was applied to each instance 10 times. The results are summarized in Tables 2 and 3. Table 2 reports the values of the best solutions found by the different methods ($Z*$), while Table 3 provides a comparison of the optimality gaps and the computational times. The formula used to calculate the gaps is $\%Gap = \frac{Z^* - Z_{LR}}{Z_{LR}}$, where Z_{LR} is the linear relaxation optimal value, computed using CPLEX 12.5. The time required by CPLEX is given in the last column of Table 3 (column LR). All the results reported (except the computational time of CPLEX) are the averages of the results obtained over the 10 runs. The best results obtained for each instance are indicated in bold. The name of the instances and their sizes (number of blocks (N) and number of periods (T)) are given in the first three columns of each table.

Table 2: Average values of the solutions obtained by the different solution methods on the first benchmark dataset, S1

			Z*($)					
	N	T	TS	VND	NFA	HH1	HH2	HH3
S1-C1	4,273	3	154,223,830	165,492,629	164,651,308	165,473,283	165,442,186	**165,554,172**
S1-C2	7,141	4	168,851,400	199,000,195	197,950,908	199,199,888	199,196,593	**199,257,683**
S1-C3	12,627	7	184,358,854	227,752,677	227,077,936	190,778,228	**229,131,920**	229,010,596
S1-C4	20,626	10	189,216,220	246,231,092	250,031,265	195,276,209	250,606,858	**250,845,094**
S1-C5	26,021	13	199,655,746	234,576,578	242,582,733	165,399,729	**244,166,092**	243,662,065
S1-G1	18,821	5	299,015,026	410,919,230	408,328,422	351,863,885	**411,113,417**	411,055,283
S1-G2	23,901	7	232,012,207	442,929,365	439,453,687	424,955,753	**443,491,029**	443,419,350
S1-G3	30,013	8	302,936,318	478,190,545	475,943,671	432,973,259	**479,413,511**	478,968,252
S1-G4	34,981	9	321,253,155	485,560,331	482,066,328	375,809,791	**487,117,548**	486,860,462
S1-G5	40,762	11	317,545,664	461,422,921	461,821,448	382,959,684	**466,327,117**	465,976,753

Table 3: Average optimality gaps and average computational times for the ten instances in the first benchmark dataset, S1

			Gap (%)						CPU (Minutes)						
	N	T	TS	VND	NFA	HH1	HH2	HH3	TS	VND	NFA	HH1	HH2	HH3	LR
S1-C1	4,273	3	6.93	0.13	0.63	0.14	0.16	**0.09**	4.27	1.05	3.26	1.02	**0.90**	1.06	0.23
S1-C2	7,141	4	15.40	0.29	0.82	0.19	0.19	**0.16**	9.52	3.26	8.09	**1.64**	2.63	2.60	5.68
S1-C3	12,627	7	19.84	0.98	1.27	17.05	**0.38**	0.43	29.46	13.00	19.29	**3.85**	5.26	4.38	139.01
S1-C4	20,626	10	25.57	3.14	1.65	23.19	1.42	**1.33**	68.76	43.95	45.20	**7.23**	21.27	11.84	1540.61
S1-C5	26,021	13	19.08	4.92	1.68	32.96	**1.04**	1.24	112.76	72.91	55.06	**7.97**	24.73	17.68	3470.63
S1-G1	18,821	5	27.49	0.35	0.98	14.67	**0.30**	0.32	31.37	17.34	78.97	**14.04**	28.12	24.78	187.77
S1-G2	23,901	7	47.93	0.59	1.37	4.63	**0.47**	0.48	55.78	31.92	100.71	**24.36**	35.00	26.20	323.75
S1-G3	30,013	8	37.10	0.71	1.18	10.10	**0.46**	0.55	80.04	57.29	168.68	**27.01**	44.82	40.59	2459.00
S1-G4	34,981	9	34.41	0.86	1.57	23.27	**0.54**	0.59	104.95	80.46	188.84	**52.32**	90.54	58.50	1179.05
S1-G5	40,762	11	32.40	1.77	1.69	18.47	**0.73**	0.80	149.47	115.06	239.51	**50.60**	79.06	75.09	2394.03

The following observations can be derived from Tables 2 and 3:

- In terms of solution quality, HH2 and HH3 outperform HH1 and the other three methods previously proposed in the literature. On average, the optimality gaps for HH2 and HH3 are 0.57% and 0.60%, respectively compared to 14.47% for HH1, 26.61% for TS, 1.37% for VND, and 1.28% for NFA.
- HH2 and HH3 are comparable in terms of solution quality, but HH3 requires slightly less computational times (on average, 26.27 minutes as opposed to 33.23 minutes). Both methods are faster than TS (64.64 minutes on average), VND (43.62 minutes on average), and NFA (90.76 minutes on average).
- Among the 6 methods, HH1 is the one that requires the least computational time (19 minutes on average), but it was not successful in solving large instances.
- As expected, all 6 methods outperform CPLEX in terms of solution time. The differences are more pronounced as the size of the instances increases.

4.2 Results for the second set of benchmark instances

In this section, we report the results obtained for the second set of benchmark instances, S2. The instances in this set

are larger than the instances in the first set and also more difficult to solve (Lamghari and Dimitrakopoulos, 2016a). The results are summarized in Tables 4 and 5, which have the same structure as Tables 2 and 3, respectively. In these tables, a dash "-" indicates that CPLEX was not able to solve the linear relaxation of the problem within the time limit (4 weeks), and thus neither the computational time of CPLEX nor the linear relaxation optimal value used to compute the gap are known.

From Table 4, it appears that among the 6 methods, HH1 is the one that provides the worst results. Moreover, its performance is far inferior to the other methods. HH2 is not as good as it was for the instances in the first dataset. In particular, for the largest instance, S2-G1, HH2 is dominated by VND, NFA, and HH3; for S2-C2, it is dominated by NFA and HH3; and for the smallest instance, S2-C1, it obtains solutions slightly better but comparable to those obtained by NFA and HH3. Results in Table 5 indicate that HH3 and NFA also outperform HH2 in terms of solution time. On average, HH2 runs 2.5 and 1.4 times longer than do HH3 and NFA, respectively. As was the case for the instances in the first dataset, HH1 is the fastest method. However, its short computational times do not compensate for the relatively poor quality of the solutions it provides. We can then conclude that, for the instances in the second benchmark dataset, regarding both solution quality and solution time, the new hyper-heuristic HH3 seems to be the best choice.

Table 4: Average values of the solutions obtained by the different solution methods on the second benchmark dataset, S2

			Z* ($)					
	N	**T**	**TS**	**VND**	**NFA**	**HH1**	**HH2**	**HH3**
S2-C1	14,118	6	19,636,316	26,951,280	27,717,050	17,459,273	**27,742,072**	27,678,106
S2-C2	28,154	16	175,562,546	211,541,744	224,243,744	111,268,140	222,996,672	**224,323,036**
S2-G1	48,821	14	333,885,325	446,586,205	469,765,792	174,228,176	402,188,005	**471,381,554**

Table 5: Average optimality gaps and average computational times for the three instances in the second benchmark dataset, S2

			Gap (%)						CPU (Minutes)						
	N	**T**	**TS**	**VND**	**NFA**	**HH1**	**HH2**	**HH3**	**TS**	**VND**	**NFA**	**HH1**	**HH2**	**HH3**	**LR**
S2-C1	14,118	6	34.09	9.53	6.96	41.40	**6.88**	7.09	28.24	42.94	27.46	**4.49**	11.61	13.07	932.00
S2-C2	28,154	16	-	-	-	-	-	-	150.16	223.07	74.43	**8.38**	51.18	40.76	-
S2-G1	48,821	14	-	-	-	-	-	-	227.84	252.56	243.64	**58.03**	410.24	136.79	-

4.3 Results for the third set of benchmark instances

We next compare the 6 methods on the ten instances of the third benchmark dataset, S3. The same comparison criteria as above are used; that is, the average values of the solutions obtained by each method (Table 6), as well as the average optimality gaps and computational times (Table 7). Again, the best results obtained for each instance are indicated in bold.

Some of the observations made in the previous sections can be confirmed from the results in Tables 6 and 7. First, all 6 methods solve the problems in a very reasonable time, in the order of a few minutes, which is significantly smaller than the 36.44 hours that CPLEX requires on average to solve the linear relaxation. Second, although HH1 is the fastest method, requiring 7.61 minutes on average, the quality of the solutions obtained with this hyper-heuristic is far from the quality obtained by the other methods. On average, the optimality gap for HH1 is 43.35% as opposed to 13.15%, 4.13%, 1.70%, 0.81%, and 1.02% for TS, VND, NFA, HH2, and HH3, respectively. HH2 and HH3 perform better than do the other methods, obtaining solutions very close to optimality, as can be seen from the small values of the gaps. When comparing these two hyper-heuristics, it appears that HH2 reaches slightly better solutions than does HH3, but it requires slightly more computational time (average times are 23.33 and 17.77 minutes for HH2 and HH3, respectively).

Table 6: Average values of the solutions obtained by the methods on the third benchmark dataset, S3

			Z* ($)					
	N	**T**	**TS**	**VND**	**NFA**	**HH1**	**HH2**	**HH3**
S3-C1	22,549	12	228,321,444	242,493,693	248,498,492	83,935,248	**251,555,912**	250,129,283
S3-C2	22,388	12	224,771,619	240,539,508	246,829,312	145,062,692	**248,975,295**	247,982,137
S3-C3	22,285	12	214,564,681	240,704,518	246,612,938	143,484,757	247,039,304	**248,641,198**
S3-C4	22,302	12	217,846,604	240,001,209	245,435,806	175,070,760	**248,962,810**	248,237,888
S3-C5	21,965	11	211,982,629	246,525,911	252,350,522	180,248,250	**253,890,092**	253,414,323
S3-C6	22,246	12	227,033,349	240,482,997	246,536,398	89,316,962	**247,831,088**	247,437,335
S3-C7	22,716	12	223,665,078	241,887,794	249,101,276	181,818,044	**252,396,077**	251,522,448
S3-C8	22,529	12	220,337,305	243,777,965	250,106,451	130,638,347	**253,237,604**	252,009,598
S3-C9	22,253	12	216,507,929	244,400,981	250,292,824	177,589,529	**251,774,218**	251,256,550
S3-C10	22,720	12	208,301,681	240,699,371	247,146,615	124,755,694	**249,670,386**	249,342,299

Table 7: Average optimality gaps and average computational times for the ten instances in the third benchmark dataset, S3

			Gap (%)						CPU (MINUTES)						
	N	**T**	**TS**	**VND**	**NFA**	**HH1**	**HH2**	**HH3**	**TS**	**VND**	**NFA**	**HH1**	**HH2**	**HH3**	**LR**
S3-C1	22,549	12	9.66	4.05	1.68	66.79	**0.47**	1.03	90.20	121.81	46.01	**7.24**	22.35	17.72	3586.66
S3-C2	22,388	12	10.44	4.16	1.65	42.20	**0.79**	1.19	89.55	125.44	51.27	**6.36**	21.31	16.88	2421.27
S3-C3	22,285	12	14.52	4.11	1.76	42.84	1.59	**0.95**	89.14	115.70	49.79	**5.15**	18.13	17.10	1420.44
S3-C4	22,302	12	13.00	4.15	1.98	30.08	**0.58**	0.86	89.21	122.06	44.01	**6.46**	22.77	18.71	3518.42
S3-C5	21,965	11	17.28	3.80	1.53	29.66	**0.93**	1.11	80.54	111.33	49.67	**11.08**	23.91	16.21	1464.72
S3-C6	22,246	12	9.28	3.91	1.49	64.31	**0.97**	1.13	88.99	117.20	54.40	**4.90**	16.93	15.12	1593.52
S3-C7	22,716	12	11.82	4.63	1.79	28.32	**0.49**	0.83	90.87	121.71	49.17	**9.97**	31.97	19.44	1986.17
S3-C8	22,529	12	13.41	4.20	1.71	48.66	**0.48**	0.96	90.12	121.68	45.91	**10.10**	32.10	19.18	1878.28
S3-C9	22,253	12	14.95	3.99	1.68	30.24	**1.10**	1.30	89.01	129.62	29.09	**5.69**	17.06	17.78	1807.55
S3-C10	22,720	12	17.17	4.28	1.72	50.39	**0.71**	0.85	90.88	121.02	51.09	**9.12**	24.42	19.53	2184.68

4.4 Results for the fourth set of benchmark instances

Finally, we compare the 6 methods on the instances of the fourth benchmark dataset, S4, which are larger than those in the third dataset, S3. Tables 8 and 9 summarize this comparison.

Although HH2 and HH3 perform similarly on the instances of S3 (c.f. previous section), for the larger instances in S4, the differences between the two hyper-heuristics are more pronounced. HH3 outperforms HH2 both in terms of solution quality and solution time. In particular, for the instance S4-C7, HH3 is significantly better than HH2, improving the value of the objective function by 24.07%. For the other 9 instances, the solutions found by HH3 are comparable to or better than those produced by HH2, very close to optimality, with an average gap of 0.36%. In general, HH2 finds better solutions than NFA, but NFA provides more consistent results in addition to being relatively faster (average solution times are 112.37 and 166.73 for NFA and HH2, respectively). Both HH2 and NFA obtain significantly better solutions than VND, TS, and HH1. The latter again gives very poor results and ranks last in terms of solution quality, but first in terms of solution time.

Table 8: Average values of the solutions obtained by the different solution methods on the fourth benchmark dataset, S4

			Z* ($)					
	N	T	TS	VND	NFA	HH1	HH2	HH3
S4-C1	40,090	21	187,240,274	193,293,330	207,895,000	55,841,661	209,773,023	**210,963,617**
S4-C2	40,090	21	187,667,839	191,649,430	206,154,000	107,754,109	209,431,022	**209,677,364**
S4-C3	40,090	21	185,647,986	192,529,050	206,859,000	108,982,799	209,353,576	**209,897,938**
S4-C4	40,090	21	187,590,230	190,326,879	206,050,000	89,424,434	207,193,028	**209,285,381**
S4-C5	40,090	21	185,890,808	190,465,025	205,583,000	51,185,270	207,322,888	**208,361,121**
S4-C6	40,090	21	181,784,804	191,922,194	206,324,000	35,729,883	209,423,226	**209,476,860**
S4-C7	40,090	21	187,069,039	192,783,890	208,036,000	88,850,992	170,282,855	**211,262,172**
S4-C8	40,090	21	192,459,132	194,323,896	209,326,000	55,019,252	208,418,071	**212,170,908**
S4-C9	40,090	21	181,557,188	195,070,133	209,333,000	91,733,866	212,080,454	**213,040,675**
S4-C10	40,090	21	186,477,431	191,091,037	206,526,000	99,422,873	209,505,997	**209,628,748**

Table 9: Average optimality gaps and average computational times for the ten instances in the fourth benchmark dataset, S4

			Gap (%)						CPU (Minutes)						
	N	T	TS	VND	NFA	HH1	HH2	HH3	TS	VND	NFA	HH1	HH2	HH3	LR
S4-C1	40,090	21	11.56	8.71	1.81	73.63	0.92	**0.36**	280.64	468.67	117.93	**18.28**	122.36	109.48	3586.66
S4-C2	40,090	21	10.80	8.90	2.01	48.78	0.45	**0.33**	280.64	473.36	108.75	**28.65**	116.44	108.54	2421.27
S4-C3	40,090	21	11.91	8.64	1.84	48.29	0.66	**0.40**	280.65	469.24	109.24	**33.22**	178.15	129.07	1420.44
S4-C4	40,090	21	10.66	9.35	1.86	57.41	1.32	**0.32**	280.64	472.00	116.84	**22.13**	156.54	120.87	3518.42
S4-C5	40,090	21	11.07	8.88	1.65	75.51	0.82	**0.32**	280.64	470.74	109.91	**13.49**	185.60	124.98	1464.72
S4-C6	40,090	21	13.55	8.73	1.88	83.01	0.40	**0.38**	280.64	469.23	113.41	**20.33**	184.66	111.92	1593.52
S4-C7	40,090	21	11.78	9.09	1.89	58.10	19.70	**0.37**	280.64	473.44	112.12	**35.50**	328.46	104.19	1986.17
S4-C8	40,090	21	9.61	8.74	1.69	74.16	2.12	**0.35**	280.64	474.70	113.26	**23.53**	109.06	101.13	1878.28
S4-C9	40,090	21	15.08	8.76	2.09	57.09	0.81	**0.36**	280.64	466.37	112.54	**29.21**	142.17	122.10	1807.55
S4-C10	40,090	21	11.41	9.22	1.89	52.77	0.47	**0.41**	280.63	467.51	109.70	**29.55**	143.90	103.95	2184.68

5 Conclusions

Mineral value chain optimization involves solving very large stochastic mixed-integer programming problems. To efficiently address these problems, this paper investigated three hyper-heuristic approaches and applied them to two different mineral value chains with different components. Hyper-heuristics offer a practical alternative to the problem-specific state-of-the art search methodologies, as they operate on a search space of heuristics rather than a search space of problem solutions, and thus are more generally applicable to a variety of problems.

The three proposed hyper-heuristic approaches fall under the category of perturbative hyper-heuristics with online learning; that is, they use a set of simple perturbative low-level heuristics to improve a candidate solution, and a score-based learning mechanism is used to decide which low-level heuristic should be applied at a given step of the search process. Two of the proposed approaches are approaches previously proposed in the literature to which some

enhancements have been introduced to improve their performance, while the third one is a novel approach that uses some of the ideas of the first two but also includes new features aimed to overcome their weaknesses. To assess the performance of the three hyper-heuristics, extensive numerical experiments were performed on 33 benchmark instances of various sizes and characteristics. The three approaches were compared to each other and to three problem-specific search methodologies from the literature; namely, a tabu search heuristic (TS), a variable neighborhood descent heuristic (VND), and a network-flow based algorithm (NFA). The major conclusions of this study are that i) the refined version of the tabu search hyper-heuristic (HH1), although being the fastest, cannot compete with any of the other 5 methods in terms of solution quality; ii) the refined version of the choice function hyper-heuristic (HH2) and the new hyper-heuristic (HH3) outperform the other methods; iii) HH2 performs as well as HH3 or slightly better on some instances, but HH3 is substantially better than HH2 on the larger and most difficult instances; iv) HH3 requires shorter computational times than do HH2, TS, VND, and NFA; and v) HH3 is the most robust approach, exhibiting consistent performance for different problems and instances.

We believe that hyper-heuristic approaches hold much promise. They do not require problem-specific knowledge and therefore can address different classes of problems instead of solving just one problem. In line with the work in this paper, the next step is to further explore single-point perturbative hyper-heuristics with online learning. To be more specific, machine learning, data mining, and data analytics techniques will be investigated to design new mechanisms to choose low-level heuristics. This should enable the generation of better learning schemes and thus more efficient hyper-heuristics. Developing frameworks to enable the use of multi-point-based search methodologies and metaheuristics as low-level heuristics (meta-hyper-heuristics) will also be examined. Meta-hyper-heuristics are particularly promising as they provide a more diverse and powerful set of algorithms to the high-level strategy. In a second stage of development, heuristic generation hyper-heuristics, which are approaches that create new heuristics from a set of other existing heuristics, will be investigated. The hybridization of heuristic selection and heuristic generation hyper-heuristics will also be explored. Such combination could lead to a more efficient and more robust search scheme.

Appendix A. Two-stage SMPS formulation

The model is described in detail in Lamghari and Dimitrakopoulos (2012), and we only briefly recall it here. The following notation is used to formulate the first stage of the problem:

- N: the number of blocks considered for scheduling.
- i: block index, $i = 1, \dots, N$.
- T: the number of periods over which blocks are being scheduled (horizon).
- t: period index, $t = 1, \dots, T$.
- P_i: the set of predecessors of block i; i.e., blocks that should be removed before i can be mined.
- W_i: the weight of block i (in tonnes).
- $\underline{W_t}$: lower bound on mining (minimum amount that should be mined during period t considering both *ore* and *waste* blocks).
- $\overline{W}_t$: upper bound on mining (maximum amount that can be mined during period t - mining equipment capacity).

To formulate the second stage, the following notation is used for each scenario:

- S: the number of scenarios used to model metal uncertainty.
- s: scenario index, $s = 1, \dots, S$.
- o_{is}: parameter indicating the group of block i under scenario s.
- m_{is}: the metal content of block i under scenario s.
- v_{its}: the discounted economic value of block i if mined during period t, and if scenario s occurs. If we denote by d_1 the discount rate and by p_{is} the economic value of block i under scenario s, then v_{its} is given by the following formula:

$$v_{its} = \frac{p_{is}}{(1 + d_1)^t}$$

The economic value of a block is defined as being the net profit associated with it. The net profit differs according to whether the block is ore or waste. In the first case, it is equal to the value of the metal content of the block less the mining, processing, and selling costs. In the second case, it is equal to minus the cost of mining the block. Furthermore, it is assumed that *ore* blocks are processed during the same period when they are mined and that the profit is also generated during that period.

Furthermore, for each period *t*, the following notation is used:

- $\underline{O_t}$: lower bound on processing (minimum amount of *ore* required to feed the processing plant during period *t*).
- $\overline{O_t}$: upper bound on processing (maximum amount weight of *ore* that can be processed in the plant during period *t* - processing plant capacity).
- c_t^{o-}: unit shortage cost associated with the failure to meet $\underline{O_t}$ during period *t* ($c_t^{o-} = \frac{c^{o-}}{(1+d_2)^t}$ where c^{o-} is the undiscounted unit shortage cost, and d_2 represents the risk discount rate).
- c_t^{o+}: unit surplus cost incurred if the total of ore extracted during period *t* exceeds $\overline{O}_t$ ($c_t^{o+} = \frac{c^{o+}}{(1+d_2)^t}$).
- $\underline{M_t}$: lower bound on metal production (minimum amount of metal that should be produced during period *t*).
- $\overline{M_t}$: upper bound on metal production (maximum amount of metal that can be sold during period t).
- c_t^{m-}: unit shortage cost associated with the failure to meet $\underline{M_t}$ during period *t* ($c_t^{m-} = \frac{c^{m-}}{(1+d_2)^t}$).
- c_t^{m+}: unit surplus cost incurred if the metal production exceeds $\overline{M}_t$ ($c_t^{m+} = \frac{c^{m+}}{(1+d_2)^t}$).

The variables used to formulate the problem are as follows:

- A binary variable is associated with each block *i* for each period *t*:
  $x_{it} = \begin{cases} 1 & \text{if block } i \text{ is mined during period } t, \\ 0 & \text{otherwise.} \end{cases}$
- In modeling the *processing constraints*, we use the variables d_{ts}^{o-} and d_{ts}^{o+} to denote the shortage and the surplus in the amount of *ore* mined during period *t* if scenario *s* occurs, respectively.
- Finally, the variables d_{ts}^{m-} and d_{ts}^{m+} measure the shortage and the surplus in metal production during period *t* under scenario *s*, respectively.

Note that the x_{it} are the first-stage decision variables. They are scenario-independent since they must be fixed before knowing the values of the uncertain parameters. The deviation variables d_{ts}^{o-}, d_{ts}^{o+}, d_{ts}^{m-}, and d_{ts}^{m+} are the second-stage (recourse) decision variables. Their values depend both on the realization of the uncertain parameters and on the values of the first-stage decision variables.

Using this notation, the SMPS can be modeled as follows.

$$\max \frac{1}{S}\left\{\sum_{s=1}^{S}\sum_{t=1}^{T}\sum_{i=1}^{N} v_{its}x_{it} - \sum_{s=1}^{S}\sum_{t=1}^{T}(c_t^{o-}d_{ts}^{o-} + c_t^{o+}d_{ts}^{o+} + c_t^{m-}d_{ts}^{m-} + c_t^{m+}d_{ts}^{m+})\right\} \tag{1}$$

(*M*) Subject to

$$\sum_{t=1}^{T} x_{it} \leq 1 \qquad i = 1,\ldots,N \tag{2}$$

$$x_{it} - \sum_{\tau=1}^{t} x_{p\tau} \leq 0 \qquad i = 1,\ldots,N,\ p \in P_i,\ t = 1,\ldots,T \tag{3}$$

$$\sum_{i=1}^{N} w_i x_{it} \leq \overline{W}_t \qquad t = 1,\ldots,T \tag{4}$$

$$\sum_{i=1}^{N} w_i x_{it} \geq \underline{W}_t \qquad t = 1,\ldots,T \tag{5}$$

$$\sum_{i=1}^{N} o_{is} w_i x_{it} + d_{ts}^{o-} \geq \underline{O}_t \qquad t = 1, \ldots, T\ ,\ s = 1, \ldots, S \tag{6}$$

$$\sum_{i=1}^{N} o_{is} w_i x_{it} - d_{ts}^{o+} \leq \overline{O}_t \qquad t = 1, \ldots, T\ ,\ s = 1, \ldots, S \tag{7}$$

$$\sum_{i=1}^{N} o_{is} m_{is} x_{it} + d_{ts}^{m-} \geq \underline{M}_t \qquad t = 1, \ldots, T\ ,\ s = 1, \ldots, S \tag{8}$$

$$\sum_{i=1}^{N} o_{is} m_{is} x_{it} - d_{ts}^{m+} \leq \overline{M}_t \qquad t = 1, \ldots, T\ ,\ s = 1, \ldots, S \tag{9}$$

$$x_{it} = 0 \text{ or } 1 \qquad i = 1, \ldots, \tag{10}$$

$$d_{ts}^{o-}, d_{ts}^{o+}, d_{ts}^{m-}, d_{ts}^{m+} \geq 0 \qquad t = 1, \ldots, T\ ,\ s = 1, \ldots, S. \tag{11}$$

The objective function (1) includes two elements to maximize the expected net present value of the mining operation, and to minimize the expected recourse costs incurred whenever the stochastic constraints are violated due to metal uncertainty. We assume that all scenarios have an equal probability of occurrence and hence the coefficient $\frac{1}{S}$ represents the probability that scenario *s* occurs. Constraints (2)-(5) are related to the non-stochastic constraints and thus are scenario-independent. Constraints (2) guarantee that each block *i* is mined at most once during the horizon (*reserve constraints*). The mining precedence is enforced by constraints (3). Constraints (4) and (5) ensure that the requirements on the mining levels are respected during each period of the horizon.

Constraints (6)–(9) are related to the stochastic constraints and thus are scenario-dependent. Constraints (6) and (7) are related to the requirements on the processing levels. For each scenario *s*, the target is to have the total weight of *ore* blocks mined during any period *t* in the interval $\left[\underline{O}_t, \overline{O}_t\right]$. If it is equal to a value smaller than $\underline{O}_t$ (respectively, larger than $\overline{O}_t$), then the shortage penalty cost is equal to $c_t^{O-} d_{ts}^{O-}$ (respectively, the surplus penalty cost is equal to $c_t^{O+} d_{ts}^{O+}$). Finally, constraints (8) and (9) indicate that, during any period *t*, the target is to have the metal production in the interval $\left[\underline{M}_t, \overline{M}_t\right]$. Otherwise, the shortage penalty cost is equal to $c_t^{m-} d_{ts}^{m-}$ or the surplus penalty cost is equal to $c_t^{m+} d_{ts}^{m+}$.

Appendix B. Two-stage SMPS+ formulation

The model is described in detail in Lamghari and Dimitrakopoulos (2016a), and we only briefly recall it here. The following notation is used:

Indices:

- *N*: the number of blocks considered for scheduling.
- *i*: block index, $i = 1, \ldots, N$.
- *T*: the number of periods over which blocks are being scheduled (horizon).
- *t*: period index, $t = 1, \ldots, T$.
- *P*: the number of processors. Note that since a stockpile is associated with each processor, the number of stockpiles is equal to the number of processors.
- *p*: processor index, $p = 1, \ldots, P$.
- *S*: the number of scenarios used to model metal uncertainty.
- *s*: scenario index, $s = 1, \ldots, S$.

Variables:

- 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 $t = 1, \dots, \tau - 1$ and $x_i^t = 1$ for all $t = \tau, \dots, T$. If *i* is not mined during the horizon, then $x_i^t = 0$ for all $t = 1, \dots, T$.. To simplify the notation in the rest of this section, we introduce a set of *N* dummy decision variables x_i^0 $(i = 1, \dots, N)$, each having a fixed value equal to 0.

- y_{ps}^{t+}: surplus in the amount of ore mined during period *t* that can be processed in *p* if scenario *s* occurs (i.e., the amount to send from the mine to the stockpile associated with p)
- y_{ps}^{t-}: amount of ore to take in period *t* from the stockpile associated with processor *p*, if scenario *s* occurs (i.e., the amount to send from the stockpile to the processor).
- y_{ps}^{t}: amount of ore in the stockpile associated with processor *p* at the end of period *t* under scenario *s*. It is assumed that the stockpile is empty at the beginning of the first period but might not be empty at the end of the planning horizon.

Parameters:

- π_s: probability that scenario *s* occurs, with $\sum_{s=1}^{S} \pi_s = 1$.
- $Pred(i)$: the set of predecessors of block *i*; i.e., blocks that have to be removed to have access to block *i* ($Pred(i) \subseteq \{1, \dots, N\}$)
- w_i: weight of block *i* in tonnes (tonnage).
- W^t: maximum amount of material (waste and ore) that can be mined during period *t* (mining capacity in tonnes).
- $\bar{c}$: undiscounted cost of mining a tonne of material.
- c_p: undiscounted cost of processing a tonne of ore in processor *p*.
- $\theta_{ips} = \begin{cases} 1 & \text{if } i \text{ is processed in } p \text{ under scenario } s, \\ 0 & \text{otherwise.} \end{cases}$
- Θ_p^t: maximum amount of ore that can be processed in processor *p* during period t (processing capacity of p in tonnes).
- r_{is}: undiscounted revenue of an already mined block *i* if sent immediately for processing (i.e., during the same period it is mined), and if scenario *s* occurs ($r_{is} = 0$ if i is waste block under scenario s).
- c_p^+: undiscounted cost of sending a tonne of ore to the stockpile associated with processor *p* (transportation cost plus handling cost). It is assumed that when a block arrives at the stockpile, it is mixed with the other material already there. The cost of this operation is included in c_p^+.
- c_p^-: undiscounted cost of taking a tonne of ore from the stockpile associated with processor *p* (transportation cost plus loading cost).
- $\tilde{r}_{is}$: an approximation of the undiscounted revenue to be generated if a tonne of ore in the stockpile associated with *p* is processed, and if scenario *s* occurs.
- *d*: the discount rate per period for cash flows.

The two-stage stochastic programming model can be summarized as follows:

$$\max -\sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{i=1}^{N} w_i \bar{c}(x_i^t - x_i^{t-1}) + \sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s r_{is}(x_i^t - x_i^{t-1}) - \sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{p=1}^{P} \sum_{s=1}^{S} \pi_s (\tilde{r}_{ps} + c_p^+) y_{ps}^{t+} + \sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{p=1}^{P} \sum_{s=1}^{S} \pi_s (\tilde{r}_{ps} - c_p^-) y_{ps}^{t-} \quad (1)$$

(*M*) Subject to

$$x_i^{t-1} \leq x_i^t \qquad \forall\, i,\ t \quad (2)$$

$$x_i^t \leq x_j^t \qquad \forall\, i,\ j \in Pred(i),\ t \quad (3)$$

$$\sum_{i=1}^{N} w_i (x_i^t - x_i^{t-1}) \leq W^t \qquad \forall\, t \quad (4)$$

$$\sum_{i=1}^{N} \theta_{ips} w_i (x_i^t - x_i^{t-1}) - y_{ps}^{t+} + y_{ps}^{t-} \leq \Theta_p^t \quad \forall\, t,\ p,\ s \tag{5}$$

$$y_{ps}^{t-1} + y_{ps}^{t+} - y_{ps}^{t-} = y_{ps}^t \quad \forall\, t,\ p,\ s \tag{6}$$

$$x_i^t = 0 \text{ or } 1 \quad \forall\, i,\ t \tag{7}$$

$$x_i^0 = 0 \quad \forall\, i \tag{8}$$

$$y_{ps}^{t+}, y_{ps}^{t-}, y_{ps}^t \geq 0 \quad \forall\, t,\ p,\ s. \tag{9}$$

$$y_{ps}^0 = 0 \quad \forall\, p,\ s. \tag{10}$$

The objective function (1) maximizes the NPV of the mine. It includes four rms:

1. The first term ($-\sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{i=1}^{N} w_i \bar{c}\ (x_i^t - x_i^{t-1})$) evaluates the total discounted cost of the extraction (discounted cost of the first stage solution).
2. The second term ($\sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{i=1}^{N} \sum_{s=1}^{S} \pi_s\, r_{is}\ (x_i^t - x_i^{t-1})$) gives the total expected discounted revenue generated if all the ore mined is sent directly for processing during the period in which it is mined (first type of recourse). Note that waste blocks do not contribute to this term because, as mentioned earlier, if *i* is a waste block under scenario s, then $r_{is} = 0$.
3. The third term ($-\sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{p=1}^{P} \sum_{s=1}^{S} \pi_s \left(\tilde{r}_{ps} + c_p^+\right) y_{ps}^{t+}$) gives the total expected discounted cost of sending ore to the stockpiles, including both the revenue lost because the ore is not processed in the period where it is available and the cost of transportation to the stockpiles (second type of recourse).
4. The fourth term ($\sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{p=1}^{P} \sum_{s=1}^{S} \pi_s \left(\tilde{r}_{ps} - c_p^-\right) y_{ps}^{t-}$) represents the total expected discounted net revenue to be generated from processing ore taken from the stockpiles; that is, revenue minus loading and transportation costs (third type of recourse).

Constraints (2)-(4) are scenario-independent. Constraints (2) guarantee that each block *i* is mined at most once during the horizon. The mining precedence is enforced by constraints (3). Constraints (4) impose an upper bound W^t on the amount of material (waste and ore) mined during each period *t*. Constraints (5) are related to the requirements on the processing levels and therefore are senario-dependent. They stipulate that for each scenario s and each processor *p*, if the total weight of ore blocks mined during any period *t* is greater than the processing capacity at that period, Θ_p^t, then the surplus y_{ps}^{t+} is sent to the stockpile associated with the processor *p*, inducing a penalty cost equal to $\frac{(\tilde{r}_{ps} + c_p^+)}{(1+d)^t}\ y_{ps}^{t+}$. If it is smaller than Θ_p^t and there is material in the stockpile, then an amount equal to y_{ps}^{t-} (maximum possible such that neither the capacity of the processor nor the amount available in the stockpile is exceeded) is taken from the stockpile and added to feed the processor, generating a net profit equal to $\frac{(\tilde{r}_{ps} - c_p^-)}{(1+d)^t}\ y_{ps}^{t-}$. Finally, constraints (6) balance the flow at each stockpile and are also scenario-dependent. They ensure that for each scenario *s*, at the end of any period *t*, the amount of ore in the stockpile associated with each processor *p* is equal to the amount that was in the stockpile at the end of the previous period $(t-1)$ plus the amount added to the stockpile during *t* minus the amount taken from the stockpile during *t* (i.e., the amount sent from the stockpile for processing, if any). The initial amount in the stockpile, y_{ps}^0, is assumed to be equal to 0.

References

Albor, F., Dimitrakopoulos, R. (2009) Stochastic mine design optimization based on simulated annealing: Pit limits, production schedules, multiple orebody scenarios and sensitivity analysis. IMM Transactions, Mining Technology, 118(2) : 80–91.

Albor, F., Dimitrakopoulos, R. (2010) Algorithmic approach to pushback design based on stochastic programming: Method, application and comparisons. IMM Transactions, Mining Technology, 119(2) : 88–101.

Asad, M., Dimitrakopoulos, R. (2013) Implementing a parametric maximum flow algorithm for optimal open pit mine design under uncertain supply and demand. Journal of the Operational Research Society, 64 : 185–197.

Behrang, K., Hooman, A., Clayton, D. (2014) A linear programming model for long-term mine planning in the presence of grade uncertainty and a stockpile. International Journal of Mining Science and Technology, 24 : 451–459.

Bienstock, D., Zuckerberg, M. (2010) Solving LP relaxations of large-scale precedence constrained problems. Lecture Notes in Computer Science, 6080 : 1–14.

Birge, J., Louveaux, F. (2011) Introduction to stochastic programming, Second Edition. Springer.

Bley, A., Boland, N., Fricke, C., Froyland, G. (2010) A strengthened formulation and cutting planes for the open pit mine production scheduling problem. Computers & Operations Research, 37(9) : 1641–1647.

Boland, N., Dumitrescu, I., Froyland, G., Gleixner, A. M. (2009) LP-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. Computers & Operations Research, 36 : 1064–1089.

Boucher, A., Dimitrakopoulos, R. (2009) Block simulation of multiple correlated variables. Mathematical Geosciences, 41(2) : 215–237.

Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R. (2013) Hyperheuristics: A survey of the state of the art. Journal of the Operational Research Society, 64 : 1695–1724.

Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S. (2003a) Hyperheuristics: An emerging direction in modern search technology. In: Handbook of Metaheuristics, Glover F and Kochenberger G (eds), Kluwer: 457–474.

Burke, E., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J. (2010) A classification of hyper-heuristic approaches. In: Handbook of metaheuristics - Second Edition, Springer, New York: 449–468.

Burke, E., Kendall, G., Soubeiga, E. (2003b) A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics, 9(3) : 451–470.

Caccetta, L., Hill, S. (2003) An application of branch and cut to open pit mine scheduling. Journal of Global Optimization, 27(2-3) : 349–365.

Chanda, E. (2007) Network linear programming optimization of an integrated mining and metallurgical complex. In: Proceedings of Orebody Modelling and Strategic Mine Planning: Uncertainty and risk management models, The Australasian Institute of Mining and Metallurgy Spectrum Series 14, 2nd Edition: 149–155.

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

Chiles, J., Delfiner, P. (2012) Geostatistics: Modeling Spatial Uncertainty, Second ed. John Wiley & Sons., New Jersey.

Cowling, P., Kendall, G., Soubeiga, E. (2001) A hyperheuristic approach for scheduling a sales summit. Lecture Notes in Computer Science, 2079 : 176–190.

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

Denzinger, J., Fuchs, M., Fuchs, M. (1997) High performance ATP systems by combining several AI methods. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97), Morgan Kaufmann, CA, USA: 102–107.

Dimitrakopoulos, R. (2011) Stochastic optimization for strategic mine planning: A decade of developments. Journal of Mining Science, 47(2) : 138–150.

Dimitrakopoulos, R., Farrelly, C., Godoy, M. (2002) Moving forward from traditional opti- mization: grade uncertainty and risk effects in open pit mine design. IMM Transactions, Mining Technology, 111(1) : A82-A88.

Dowd, P. (1994) Risk assessment in reserve estimation and open-pit planning. Transactions of the Institution of Mining and Metallurgy, 103 : A148-A154.

Drake, J., Ozcan, E., Burke, E. (2012) An improved choice function heuristic selection for cross domain heuristic search. Lecture Notes in Computer Science, 7492 : 307–316.

Ferland, J. A., Amaya, J., Djuimo, M. S. (2007) Application of a particle swarm algorithm to the capacitated open pit mining problem. In: Autonomous Robots and Agents, Mukhopadhyay S. and Sen Gupta G. Ed. Springer-Verlag: 127–133.

Fisher, H., Thompson, G. (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Industrial Scheduling, Muth, J.F., Thompson, G.L. (eds.), Prentice- Hall, New Jersey: 225–251.

Fricke, C., Velletri, P., Wood, R. (2014) Enhancing risk management in strategic mine planning through uncertainty analysis. In: Proceedings of Orebody Modelling and Strategic Mine Planning Symposium 2014, The Australasian Institute of Mining and Metallurgy: 275–279.

Godoy, M. (2002) The effective management of geological risk. Ph.D. thesis, University of Queensland, Australia.

Goodfellow, R., Dimitrakopoulos, R. (2014) Stochastic optimisation of mineral value chains- Developments and applications for the global optimisation of mining complexes with uncertainty. In: Orebody Modelling and Strategic Mine Planning, Ed. The Australasian Institute of Mining and Metallurgy: 13–21.

Goovaerts, P. (1997) Geostatistics for Natural Resources Evaluation. Oxford University Press, New York.

Hochbaum, D., Chen, A. (2000) Improved planning for the open-pit mining problem. Operations Research, 48(6) : 894–914.

Hoerger, S., Hoffman, L., Seymour, F. (1999) Mine planning at Newmont's Nevada operations. Mining engineering, 51(10) : 26–30.

Horta, A., Soares, A. (2010) Direct sequential co-simulation with joint probability distributions. Mathematical Geosciences, 42(3) : 269–292.

Kawahata, K., Schumacher, P., Fein, M. (2015) Strategic mine planning and production scheduling optimization at Newmont's twin creeks operation. In: Proceedings of the 37th International Symposium Application of Computers and Operations Research in the Mineral Industry (APCOM), Fairbanks, Alaska: 1052–1060.

Lamghari, A., Dimitrakopoulos, R. (2012) A diversified tabu search approach for the open- pit mine production scheduling problem with metal uncertainty. European Journal of Operational Research, 222(3) : 642–652.

Lamghari, A., Dimitrakopoulos, R. (2016a) Network-flow based algorithms for scheduling production in multi-processor open-pit mines accounting for metal uncertainty. European Journal of Operational Research, 250(1) : 273–290.

Lamghari, A., Dimitrakopoulos, R. (2016b) Progressive hedging applied as a metaheuristic to schedule production in open-pit mines accounting for reserve uncertainty. European Journal of Operational Research, 253(3) : 843–855.

Lamghari, A., Dimitrakopoulos, R., Ferland, J.A. (2015) A hybrid method based on linear programming and variable neighborhood descent for scheduling production in open-pit mines. Journal of Global Optimization, 63(3) : 555–582.

Lamghari, A., Dimitrakopoulos, R., Ferland, J.A. (2014) A variable descent neighborhood algorithm for the open-pit mine production scheduling problem with metal uncertainty. Journal of the Operational Research Society, 65(9) : 1305–1314.

Maleki, M., Emery, X. (2015) Joint simulation of grade and rock type in a stratabound copper deposit. Mathematical Geosciences, 47(4) : 471–495.

Marcotte, D., Caron, J. (2013) Ultimate open pit stochastic optimization. Computers & Geosciences, 51 : 238–246.

Menabde, M., Froyland, G., Stone, P., Yeates, G. (2007) Mining schedule optimization for conditionally simulated orebodies. Orebody Modelling and Strategic Mine Planning, The Australasian Institute of Mining and Metallurgy, Spectrum Series, 14 : 379–384.

Montiel, L., Dimitrakopoulos, R. (2015) Optimizing mining complexes with multiple processing and transportation alternatives: An uncertainty-based approach. European Journal of Operational Research, 247(1) : 166–178.

Montiel, L., Dimitrakopoulos, R., Kawahata, K. (2016) Globally optimising open-pit and underground mining operations under geological uncertainty. Mining Technology, 125(1), 2–14.

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

Ramazan, S., Dimitrakopoulos, R. (2013) Production scheduling with uncertain supply: A new solution to the open pit mining problem. Optimization and Engineering, 14 : 361–380.

Ravenscroft, P. (1992) Risk analysis for mine scheduling by conditional simulation. Transactions of the Institution of Mining and Metallurgy, Section A: Mining Technology: A104–A108.

Ross, P. (2005) Hyper-heuristics. In: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Burke EK and Kendall G (eds), Springer: 529–556.

Rossi, M., Deutsch, C. (2014) Mineral Resource Estimation. Springer, New York.

Whittle, G. (2007) Global asset optimization. In: Proceedings of Orebody Modelling and Strategic Mine Planning: Uncertainty and risk management models, The Australasian Institute of Mining and Metallurgy Spectrum Series 14, 2nd Edition: 331–336.

Whittle, J. (2009) The global optimizer works-what next? In: Proceedings of Advances in Orebody Modelling and Strategic Mine Planning: old and new dimensions in a changing world, The Australasian Institute of Mining and Metallurgy Spectrum Series 17, 1st Edition: 3–5.