Learning to Schedule Heuristics for the Simultaneous Stochastic Optimization of Mining Complexes

Yassine Yaakoubi,[123*] **Roussos Dimitrakopoulos**[12]
[1]COSMO (Stochastic Mine Planning Laboratory) - McGill University, Canada
[2]GERAD (Group for Research in Decision Analysis), Canada
[3]Mila (Montreal Institute for Learning Algorithms), Canada
{yassine.yaakoubi,roussos.dimitrakopoulos}@mcgill.ca

Abstract

The simultaneous stochastic optimization of mining complexes (SSOMC) is a large-scale combinatorial optimization problem that manages the extraction of materials from multiple mines and their processing using interconnected facilities. Following the work of Zarpellon et al. (2020) and Chmiela et al. (2021), to the best of our knowledge, this work proposes the first data-driven framework for heuristic scheduling in a hyper-heuristic-based solver that is fully self-managed to solve the SSOMC. The proposed learn-to-perturb (L2P) hyper-heuristic is a multi-neighborhood simulated annealing algorithm. The L2P selects the heuristic (perturbation) to apply in a self-adaptive manner using reinforcement learning (RL) to efficiently explore which local search is best suited for a particular search point. Several state-of-the-art agents have been incorporated into the proposed hyper-heuristic to better adapt the search and guide it towards better solutions. By learning from data describing the performance of heuristics, a problem-specific ordering of heuristics that collectively finds better solutions faster is obtained. The L2P is tested on several real-world mining complexes, with an emphasis on efficiency, robustness, and generalization capacity. Results show a reduction in the computational time by 30-45%.

Introduction

Solving a large-scale industrial mining complex problem is critical for decision-makers in the mining industry. An industrial mining complex is an engineering system that manages the extraction of materials from multiple mines, their processing in interconnected facilities, and multiple waste sites to generate a set of products delivered to the customers. Mine scheduling optimization is critical as sub-optimal scheduling can adversely affect profit. Since mining complexes have multiple mining sites, the material must be extracted from the sites simultaneously. This requires the simultaneous optimization of mining complexes (Goodfellow and Dimitrakopoulos 2016). The main objective of the mine production scheduling problem is to find an approach to extract the profitable ore blocks (e.g., above a cut-off grade).

Conventional approaches to solving these problems break down the optimization into separate, sequential, linear optimization steps, leading to sub-optimal solutions, ignoring the various nonlinear interactions between processing, transportation, and destination policies that also contribute to the value of mined products. The simultaneous stochastic optimization of mining complexes (SSOMC) overcomes these limitations by incorporating all components of the mining complexes into a single mathematical model. The model maximizes the Net Present Value (NPV) of the mining complexes and includes geological uncertainty to manage the associated risk. As exact methods are impractical to use for the SSOMC with more than a few thousand blocks, researchers have also employed meta-heuristics and hyper-heuristics to address these large-scale global optimization problems.

Meta-heuristics are problem-independent algorithms that use a search algorithm to find a near-optimal solution by developing heuristics (Talbi 2009; Glover and Kochenberger 2006). They combine these (problem-specific) heuristics in a more general framework, as in Khan and Niemann-Delius (2015) where the authors use particle swarm optimization (PSO) to schedule open-pit mines. Most notably, a simulated annealing approach was proposed by Goodfellow and Dimitrakopoulos (2016), resulting in COSMO Suite (Goodfellow and Dimitrakopoulos 2019), a software package capable of integrating the various components of the mining complexes and solving the optimization problem in a reasonable time. However, early performance might dominate recent performance in that the score function only uses a combination of past performance to sample the perturbations to be applied.

A recent development to meta-heuristic optimization is hyper-heuristics. For example, Chatterjee and Dimitrakopoulos (2019) combine several existing methods to find a solution. Lamghari, Dimitrakopoulos, and Senécal (2021) combine machine learning (ML), exact algorithms, and heuristic methods to solve the SSOMC. The authors show that using neural networks as a surrogate model for the downstream sub-problem instead of solving it to optimality reduces computational time significantly.

Machine learning has shown promise in tackling small-scale operations research (OR) tasks such as using graph convolutional neural networks to learn branch-and-bound variable selection policies (Gasse et al. 2019) and using hybrid architectures for efficient branching on CPU machines (Gupta et al. 2020), as well as guiding OR algorithms on large-scale tasks such as using neural networks (Yaakoubi, Soumis, and Lacoste-Julien 2019, 2020) and struc-

[*]Corresponding author.

tured convolutional kernel networks (Yaakoubi, Soumis, and Lacoste-Julien 2021) to solve airline crew scheduling (Yaakoubi 2019). Furthermore, recent research goes a step further by proposing data-driven methods that can guide OR algorithms to solve combinatorial optimization problem. Indeed, given the complexity of these problems, state-of-the-art algorithms rely on specific heuristics for making decisions that are otherwise computationally expensive or mathematically not well-defined. As a result, ML presents itself as a fitting predictive tool to make such heuristic-scheduling decisions in a more principled and optimized way (Bengio, Lodi, and Prouvost 2020). While Zarpellon et al. (2020) presents a novel imitation learning framework and hypothesize that parameterizing the state of the Branch-and-Bound search tree can aid in solving mixed-integer linear programming problems (MILPs), Chmiela et al. (2021) propose the first data-driven framework for scheduling heuristics in an exact mixed-integer programming (MIP) solver, where they obtain a problem-specific schedule of heuristics that collectively find many solutions at minimal cost by learning from data describing the performance of primal heuristics. The authors are able to significantly outperform the then state-of-the-art methods for "learning to branch" by allowing better generalization to unseen problems.

We follow this line of work by proposing, to the best of our knowledge, the first data-driven framework for heuristic scheduling in a hyper-heuristic-based solver that is fully self-managed. Indeed, in most of the literature on the use of data-driven AI for OR methods, authors assume structural knowledge of the scheduling problem (e.g., optimal solutions, decisions made by strong branching, etc.). This assumption allows them to apply ML to provide fast approximations to these decisions. Little research has been done to solve large-scale combinatorial optimization problems such as the SSOMC where no structural knowledge pre-exists. In this case, ML models can be deployed to discover the structure of the problem. This is called *policy learning* and is an application of reinforcement learning (RL). In the current manuscript, the use of hyper-heuristics is justified by the extremely large size and complex constraints of the SSOMC and the demonstrated success of simulated annealing (Metropolis et al. 1953; Kirkpatrick, Gelatt, and Vecchi 1983) and hyper-heuristics (Godoy 2003; Kumral 2013) in solving the SSOMC. Indeed, following the work of Goodfellow and Dimitrakopoulos (2016, 2017); Lamghari, Dimitrakopoulos, and Senécal (2021), a new approach is proposed to improve the search strategy by using past experience to better guide the exploration. A self-learning hyper-heuristic called L2P (learn-to-perturb) is proposed, using a multi-neighborhood simulated annealing algorithm, where the selection of a perturbation is self-adaptive using RL. By defining a neighborhood structure, the RL agent learns how to guide the search over the solution space landscape to find better solutions during the optimization process.

The paper makes the following novel contributions: (1) proposing a self-managed hyper-heuristic for solving the SSOMC based on a multi-neighborhood simulated annealing algorithm with adaptive neighborhood search; (2) using RL to adapt the search and guide it towards better solutions; (3) testing on several real-world mining complexes with emphasis on efficiency, robustness, and generalization capacity. To test the proposed methodology, three policy-based agents are proposed: advantage actor-critic (A2C) (Mnih et al. 2016), proximal policy optimization (PPO) (Schulman et al. 2017), and soft actor-critic (SAC) (Haarnoja et al. 2018). Results demonstrate L2P's effectiveness on a range of real-world mining complexes, where the proposed solution process takes only minutes, while CPLEX solves the linear relaxation in days or weeks. Using RL further reduces the computational time by 30-45%

Problem Statement

A mathematical formulation of the SSOMC motivated from Lamghari and Dimitrakopoulos (2016) and Goodfellow and Dimitrakopoulos (2016) is briefly discussed, along with the implementation details of COSMO Suite, which the proposed L2P hyper-heuristic is implemented on. The objective function maximizes the expected NPV of the mining complex (total expected discounted profit generated from processing ore) and minimizes the expected recourse costs incurred whenever the stochastic (scenario-dependent) constraints are violated. The solving of the SSOMC needs to take into account the deterministic (scenario-independent) and stochastic (scenario-dependent) constraints.

The scenario-independent constraints include:

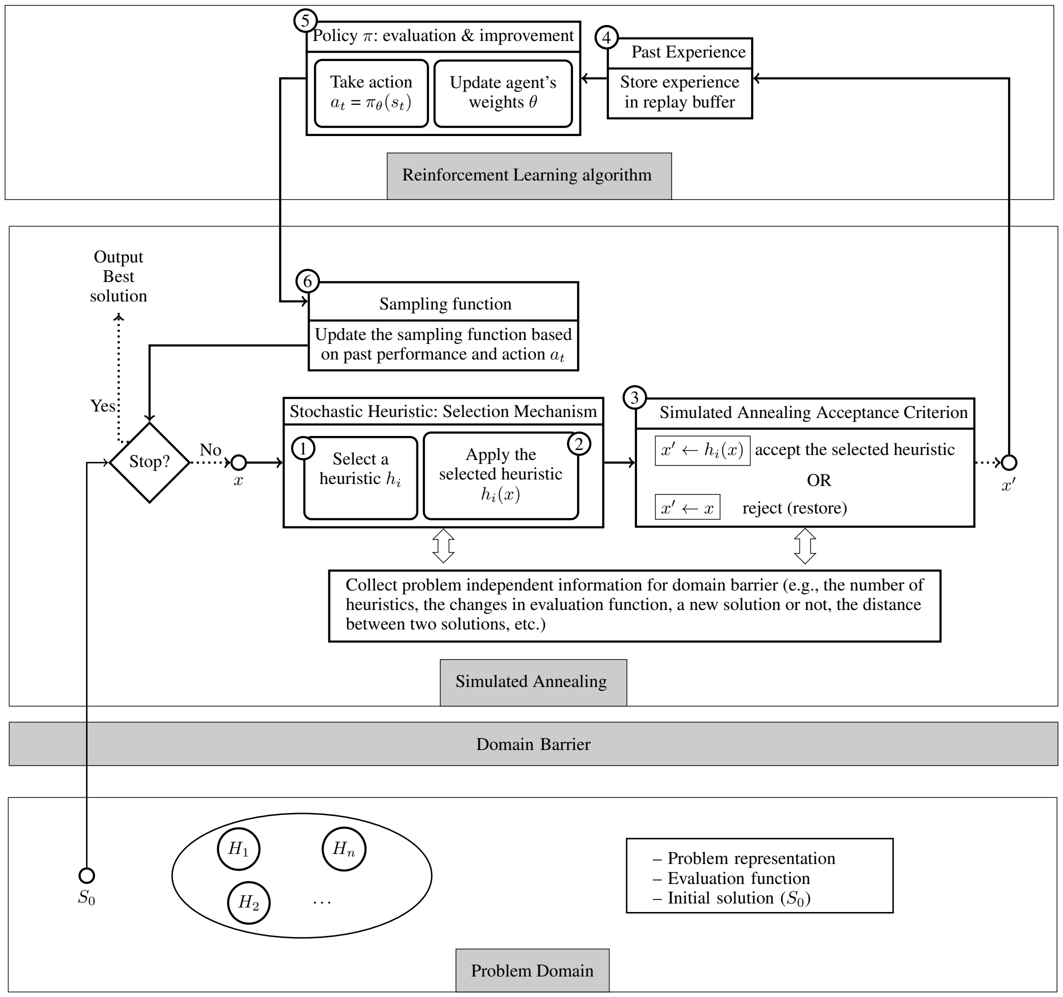
- **Reserve constraints** ensure that each block is mined at most once during the horizon.
- **Slope constraints** guarantee that each block can only be mined after all its predecessor blocks have been mined.
- **Mining constraints** limits the maximum rate of mining.

The scenario-dependent constraints include:

- **Processing constraints** impose a minimum and maximum amount of material to be processed by each processor per period.
- **Stockpiling constraints** balance the flow at each stockpile and impose a maximum amount of material of ore to be stockpiled for each stockpile per period.

We build the proposed L2P approach into the COSMO Suite framework. More specifically, Goodfellow and Dimitrakopoulos (2016, 2017) proposed a generic modeling approach resulting in COSMO Suite (Goodfellow and Dimitrakopoulos 2019) to overcome the extreme size that the model can reach and allow the modeling of varying mining complexes. In COSMO Suite, a "material" describes a product extracted from a mine or generated via blending, separation, or processing. An "attribute" describes a material's property of interest. Attributes may be primary (variables are passed from one location to another, e.g., the metal tonnage) or hereditary (not forwarded between locations, e.g., costs). In COSMO Suite, (non-)linear equations can be assigned to these hereditary attributes and are evaluated dynamically during optimization. The simultaneous stochastic optimizer can modify three types of decision variables:

1. **Production scheduling decisions** define whether or not a certain block is extracted in a certain period.



5
Policy π: evaluation & improvement
Take action $a_t = \pi_\theta(s_t)$
Update agent's weights θ
4
Past Experience
Store experience in replay buffer
Reinforcement Learning algorithm
Output Best solution
6
Sampling function
Update the sampling function based on past performance and action a_t
Yes
Stop?
No
x
Stochastic Heuristic: Selection Mechanism
1
Select a heuristic h_i
2
Apply the selected heuristic $h_i(x)$
3
Simulated Annealing Acceptance Criterion
$x' \leftarrow h_i(x)$ accept the selected heuristic
OR
$x' \leftarrow x$ reject (restore)
x'
Collect problem independent information for domain barrier (e.g., the number of heuristics, the changes in evaluation function, a new solution or not, the distance between two solutions, etc.)
Simulated Annealing
Domain Barrier
S_0
H_1
H_n
H_2
...
– Problem representation
– Evaluation function
– Initial solution (S_0)
Problem Domain

Figure 1: Illustration of the learn-to-perturb (L2P) hyper-heuristic.

2. **Destination policy decisions** decide where each group/-cluster (of similar grade) is sent per period. More specifically, as an extension to the robust cut-off grade policies, multivariate distribution clustering divides blocks into groups based on multiple elements (e.g., grades) within each material type. The optimizer is tasked with deciding where each group (cluster) is sent per period.
3. **Processing stream decisions** define the proportion of a product sent from a location to a destination.

Proposed Methodology

This section presents the learn-to-perturb (L2P) solution approach that employs simulated annealing, Tabu search (Glover and Laguna 1998), and RL to solve the SSOMC. The proposed self-learning hyper-heuristic is constructed by hybridizing a multi-neighborhood simulated annealing algorithm with RL. A probabilistic approach works towards identifying a global optimum by identifying neighboring states with a lower energy state or nearer to an optimal solution over the solution space landscape.

General Structure

Rather than solving an optimization problem directly, the hyper-heuristic tries to identify the best heuristic for a given problem state and compensate for various heuristic methods' weaknesses through simultaneous optimization. Indeed, while a set of low-level heuristics (perturbations) work on finding optimality simultaneously, a high-level heuristic schedules their sequencing (ordering), ensuring that the most optimal solution is found by combining the perturbations. The proposed hyper-heuristic is illustrated in Figure 1 and is divided into three components that are used simultaneously (rather than in sequential order):

- **Problem domain:** includes the problem representation, evaluation function, initialization procedure, and set of low-level heuristics. This is delineated in the bottom block of Figure 1.
- **Simulated annealing algorithm:** composed of a stochastic heuristic selection mechanism, an acceptance criterion, and a sampling function.
- **RL algorithm:** to predict the heuristics' future performance to guide the search towards better solutions.

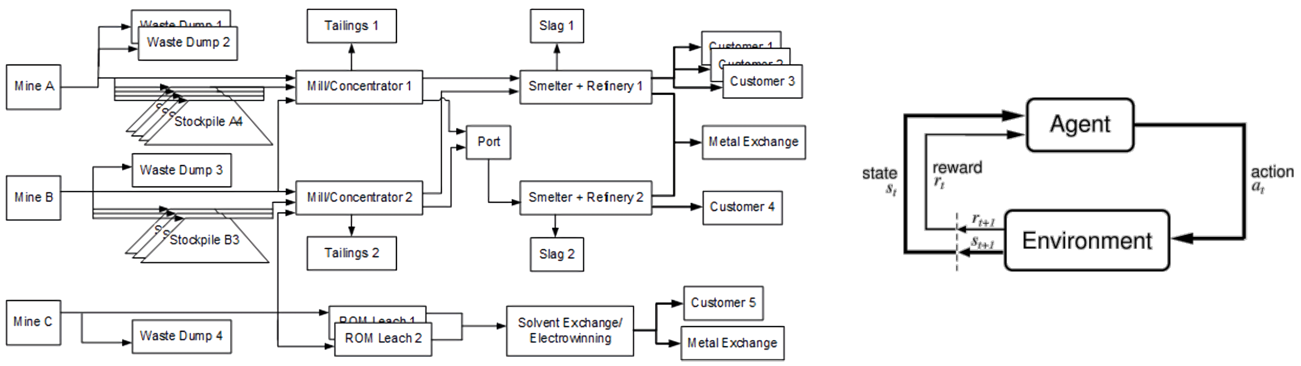


Figure 2: The left plot shows an illustration of a typical mining complex. The right plot shows an interaction between the RL agent and the environment (Sutton and Barto 2018).

As in boxes 1-3 of Figure 1, at each iteration, the approach attempts to improve the current solution x by iterating between heuristic selection and move acceptance. More specifically, once a heuristic is selected (see box 1), a candidate solution x is modified into a new solution using said heuristic (see box 2). The heuristic is chosen based on its past performance and the RL agent's predicted performance. Simulated annealing handles the heuristic selection mechanism and the acceptance criterion.

The algorithm begins with one state and then moves to a neighboring state in search of a better solution. This process is repeated until no significant improvement is possible. Each time a heuristic h_i ($i = 1, \ldots, n$, where n is the number of low-level heuristics), is used the algorithm returns the resulting change in the objective function value ($\Delta f(h_i)$ as well as the time required $(T(h_i))$. Heuristics are all tested once in the first stage, and $\Delta f(h_i)$ and $(T(h_i)$ are used to calculate the initial score function $S_1(h_i)$ as in Eq. (1).

$$S_1(h_i) = \begin{cases} \dfrac{\Delta f(h_i)}{T(h_i)} & \text{if } \Delta f(h_i) > 0 \\ \dfrac{1}{\Delta f(h_i) T(h_i)} & \text{otherwise.} \end{cases} \tag{1}$$

In the second stage, the heuristic selection is performed based on the heurisitcs' tabu status and performance. Each time, a heuristic is chosen. If it does not improve the current solution, it becomes tabu and is temporarily removed from the selection set. Rather than using a fixed-length tabu list, the heuristic is made tabu for γ_T iterations where γ_T is randomly generated in the interval $[\Gamma_{\min}, \Gamma_{\max}]$. Each non-tabu heuristic h_i is associated with a selection probability p_i computed by normalizing the score function $p_i = \frac{S_F(h_i)}{\sum_{k:H_k} S_F(H_k)}$. As in box 6 of Figure 1, the score function is updated every ζ iterations, and two measures $\pi_1(h_i)$ and $\pi_2(h_i)$ are calculated for each heuristic h_i, as in Eq. (2)-(3), respectively.

$$\pi_1(h_i) = \begin{cases} \pi_1(h_i) + \dfrac{\Delta f(h_i)}{T(h_i)} & \text{if } \Delta f(h_i) > 0 \\ \pi_1(h_i) & \text{otherwise.} \end{cases} \tag{2}$$

$$\pi_2(h_i) = \begin{cases} \pi_2(h_i) + \dfrac{1}{\Delta f(h_i) T(h_i)} & \text{if } \Delta f(h_i) < 0 \\ \pi_2(h_i) & \text{otherwise.} \end{cases} \tag{3}$$

Every ζ iterations, the score function S_1 is updated using Eq. (4), where $\eta(h_i)$ is the number of times the heuristic h_i has been selected in the last ζ iterations, and α and β are two weight adjustment parameters in $[0, 1]$ defining the importance given to recent performance and to improving heuristics, respectively. The value of β is initialized to 0.5 and is modified every ζ iterations (based on whether a new best solution has been found during the last ζ iterations). All the heuristics' score measures for the last L_w rounds are provided to the RL agent as an input s_t (in the form of a $3D$ matrix). As in box 5 of Figure 1, the agent outputs the action $a_t = \pi_\theta(s_t)$, normalized and referred to as S_2, where π_θ is the agent's policy that is parameterized by the weights θ.

$$S_1(h_i) = \begin{cases} S_F(h_i) & \text{if } \eta(h_i) = 0 \\ (1-\alpha) S_F(h_i) + \alpha \dfrac{\beta \pi_1(h_i) + (1-\beta)\pi_2(h_i)}{\eta(h_i)} & \text{otherwise.} \end{cases} \tag{4}$$

Then, a voting mechanism (box 6 of Figure 1) is used to calculate the final score function S_F based on S_1 and S_2, as in Eq. 5, where λ_{RL} is the contribution of the RL agent towards the final score function. The voting mechanism combined with sampling typically promotes exploration in RL to ensure the most viable outcome.

$$S_F(h_i) = (1 - \lambda_{\text{RL}}) S_1(h_i) + \lambda_{\text{RL}} S_2(h_i). \tag{5}$$

Finally, using the updated score function, a heuristic is selected (box 1). A new solution x' is obtained using normalized S_F (box 2). The probability of accepting x' is based on the distribution in Eq 6, where $\Delta f(h_i)$ is the difference in the objective function value before and after the perturbation h_i (between x and x', respectively).

$$p(h_i(x)) = \begin{cases} 1 & \text{if } \Delta f(h_i) > 0 \\ \exp\left(-\dfrac{\Delta f(h_i)}{\text{Temp}} \right) & \text{otherwise.} \end{cases} \tag{6}$$

Once the solution vector is updated, the change in the objective function value $\Delta f(h_i)$ and the time of execution $T(h_i)$ are calculated. The reward r for the agent taking the action S_2 is $\Delta f(h_i)$. As in box 4 of Figure 1, the agent stores past experience in a replay buffer to be used when updating its weights θ. This process is repeated until a predefined stopping condition is met and only problem-independent information flow is allowed between the problem domain and hyper-heuristic layers.

Reinforcement Learning

Reinforcement learning starts from an initial state and works its way up to an optimal policy that gives the best possible performance. In what follows, policy is referred to as π, states as s, actions as a, and rewards as r. We incorporate three state-of-the-art policy-gradient agents into the RL framework: advantage actor-critic (A2C) (Mnih et al. 2016), proximal policy optimization (PPO) (Schulman et al. 2017), and soft actor-critic (SAC) (Haarnoja et al. 2018). Policy-gradient is a robust methodology that uses RL to solve problems. By modeling and improving the policy, the end goal of this method is to follow an optimized strategy path to get maximum rewards. The policy is modeled using a parameterized function to obtain the long-term cumulative reward by exploiting the gradient descent algorithm.

At each timestamp t, the state s_t is defined as all the information (performance) related to all the perturbations (low-level heuristics) for the last L_w iterations. Each state is a $3D$ matrix of dimensions $L_w \times 3 \times n$. The first axis (of dimension L_w) corresponds to the number of iterations from which useful information is extracted (i.e., the heuristics' performance). The second axis (of dimension 3) corresponds to the number of features or measures for each heuristic. For each iteration, the normalized score functions are saved: π_1 is defined in Eq. 2, π_2 is defined in Eq. 3, and S_1 is defined in Eq. 4. The third axis (of dimension n) corresponds to the number of the low-level heuristics.

The agent outputs the normalized action ($a_t = \pi_\theta(s_t)$), where π is the agent's policy parameterized by the weight θ. We denote by π_θ the stochastic policy with parameter θ. The stochasticity comes from adding a Gaussian noise $N(0, \sigma)$ to the outputted (deterministic) action a_t to obtain the score function S_2. Although σ can be learned during training, it is used as a hyperparameter in this case to avoid an overconfident agent (early convergence of the policy).

After calculating the score function S_2, a voting mechanism is used to calculate the final score function S_F based on S_1 and S_2, as formulated in Eq. 5, where λ_{RL} denotes the RL agent's contribution to the final score function that will be used for the next optimization iteration. In the following iteration, a heuristic h_i is chosen using the calculated score function S_F, and a new (updated) solution vector is obtained, along with the resulting change in the objective function value $(\Delta f(hi)$ and the time required $(T(h_i))$. The reward for taking action a_t is defined as $r_t = \Delta f(h_i)$. Note that in order to permit generalization and make the agent instance-independent, the calculated returns are standardized (transformed to have zero mean and unit standard deviation) before using them to update the agent's policy.

Problem Domain Component

To produce new solutions, the hyper-heuristic described in the previous section uses 38 simple perturbative low-level heuristics referred to as h_i where $i = 1, \ldots, 38$. Each heuristic aims to improve the current solution and examines a subset of one of four neighborhood categories:

1. **Block extraction sequence perturbations**: For each of the heuristics in this neighborhood, a block is chosen randomly. Then any potential slope constraints that have been violated are fixed.
2. **Cluster destination policy perturbations:** A cluster is a group of blocks in a mine that can be clubbed together based on some characteristics. Clusters can be formed by aggregating blocks with similar geochemical or metallurgical parameters. In this case, multiple scenarios are optimized simultaneously, each with varying cluster characteristics. In the heuristics of this neighborhood, a cluster destination decision variable is randomly selected and sent to a different destination (Goodfellow and Dimitrakopoulos 2016).
3. **Destination policy perturbations:** These heuristics will dynamically vary the cut-off range characteristics of clusters when cluster-related perturbations happen, allowing for further optimization.
4. **Processing stream perturbations:** A random normal number $N(y_{i,j,t,s}, 0.1)$ is employed to modify a randomly selected process stream variable in this model. Keeping the variance of the normal distribution sufficiently small, it can be ensured that value can facilitate both global and local search.

Computational Experiments

To assess the efficiency and robustness of the proposed method, numerical experiments on different instances (of the SSOMC) of various sizes and characteristics are performed. The variants of the proposed hyper-heuristic (with and without RL) are compared based on the evolution of the objective function value with respect to the computational time.

Benchmark Instances and Parameter Setting

Five instances are used to test the proposed method. The first set consists of three instances (mining complexes) referred to as $I_1 - I_3$, intended to make a first assessment of the proposed L2P hyper-heuristic with a focus on the generalization capability of the proposed method. the instances are small-medium size with up to 15,000 blocks for one mine or up to 4,000 blocks per mine for two mines. I_1 contains one copper mine, one processor, and one waste dump. I_2, the mining complex contains two mines with two metal types (oxides and sulfides) two processors (oxide leach pad and sulfide processor) and one waste dump. I_3 exceeds I_2 with a larger number of blocks and a sulfide stockpile. Each time, the method is initialized, pre-trained on one of the instances, and then tested on all three. The second set consists of two large-scale mining complexes referred to as I_4 and I_5. It is intended to assess the performance of the L2P hyper-heuristic in an actual use scenario. First, the method

		$T_{\text{gap}=1\%}$ (in minutes)				LR
Testing Instance	Training Instance	L2P–Baseline	L2P–A2C	L2P–SAC	L2P–PPO	
I1	I1	18.3 (**1.2**)	**8.5** (2.4)	14.2 (1.7)	9.8 (**1.3**)	2100
	I2		10.0 (2.8)	14.8 (1.7)	11.1 (1.5)	
	I3		9.6 (3.0)	14.6 (1.9)	11.3 (1.5)	
I2	I1	65.2 (**1.9**)	32.7 (6.7)	53.4 (3.2)	38.2 (3.0)	6480
	I2		**29.3** (5.9)	51.9 (2.8)	34.1 (**2.7**)	
	I3		30.4 (5.8)	53.9 (2.9)	33.8 (**2.7**)	
I3	I1	117.8 (**3.7**)	59.2 (12.2)	96.1 (5.7)	68.9 (5.4)	8400
	I2		**52.9** (10.8)	93.8 (5.1)	61.6 (**4.8**)	
	I3		54.8 (10.5)	97.4 (5.3)	61.2 (4.9)	

Table 1: Result summary (mean and standard deviation) of the computational time $T_{\text{gap}=1\%}$ (in minutes) to achieve a solution that is 1% far from the linear relaxation

is initialized, pre-trained on I_4, and tested on the same instance. Then, the weights are transferred, and the method is tested on the instance I_5. The instance I_4 is similar to I1 except it has 14X more blocks and imposes a lower processing bound. The instance I_5 is similar to I_3 except it contains 75X more blocks. For fast execution and efficient memory and GPU allocation, while the simulation and the simulated annealing algorithm are implemented in C++, the RL agents are written in Python using the Pytorch library (Paszke et al. 2019). The experiments are carried out on a standard Windows machine with an 8-core CPU, 32 GB of RAM, and a GPU (4GB of GDDR6 memory).

Numerical Results

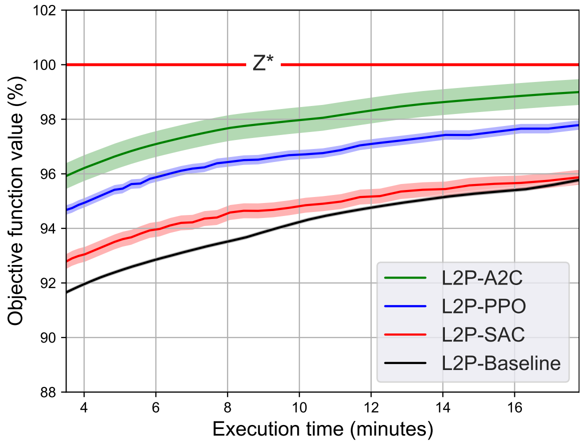
In what follows, we report numerical results for the two stages of testing. L2P–Baseline refers to the hyper-heuristic with $\lambda_{\text{RL}} = 0$ (not using RL). L2P–A2C, L2P–PPO, and L2P–SAC refer to the hyper-heuristic using A2C, PPO, and SAC as RL agents, respectively. All variants use $\lambda_{\text{RL}} = 0.5$.

First Stage of Testing To study the performance, robustness, and generalization capacity of the proposed approach, the first stage of testing is done using three instances L1-L3 where L2P is trained for one instance then trained on all three instances. Since L2P starts with a random initial solution and the heuristic selection mechanism is stochastic, all experiments are run 10 times and the following results are reported in the form of mean and standard deviation. To compare different methods, linear relaxation of the model is used, leading to a weak bound on the objective value of the optimal solution. A time limit of four weeks is set to solve the linear relaxation (LR) of each instance using CPLEX 12.10 (IBM 2020). Using the value of the obtained LR, we report in Table 1 the execution time ($T_{gap=1}$), required the variants to obtain a solution that has an objective function value 1% far from the LR value. The last row of Table 1 reports the time required by CPLEX to solve the LR. Results show that all RL-based hyper-heuristics outperform L2P–Baseline. L2P–A2C outperforms the other variants of the hyper-heuristic, reducing the execution time by 45%-55%. L2P-PPO and L2P-SAC reduce the execution time by 38%-48% and 17%-22% respectively.

Although the training is done on the heuristic space rather than the solution space, the choice of heuristic and the performance may vary slightly from one instance to another, which is a good indicator of the generalization capacity of the proposed method, especially since the weights are not updated in real-time at this stage of testing. Also, L2P–A2C is not as consistent in performance from execution to another as L2P–PPO (the second best-performing variant). The standard deviations (for $T_{gap=1\%}$) for L2P–A2C are larger than that for L2P–PPO. This may be explained by the fact that PPO makes much smaller steps to update its policy, and thus presents a more stable strategy than A2C (albeit a less performing one). Furthermore, the standard deviation of L2P–Baseline is smaller than the standard deviation of all other variants. This is because although the heuristics choice is the result of a sampling process, the score function S_1 is strictly biased towards the most-performing heuristics, and S_2 might be more biased towards exploration, especially if the state given by recent performance has not been seen before. Thus, using the score function solely S_1 can yield a more “deterministic” behavior. Due to the large size of the instances in question, CPLEX takes much more time ($\times$150-250 times) to solve the linear relaxation than any of the L2P hyper-heuristic variants take to solve the instances. In short, the best performing variant of the hyper-heuristic is L2P–A2C, both for the instance on which it was trained and for the instances on which it was not trained, and the least-performing RL–based variant is L2P–SAC.

Second Stage of Testing As the second stage of testing, the agents are trained and tested on the instance I_4. CPLEX was unable to solve the linear relaxation of the instance I_4 within the time limit (four weeks) so the objective value of the best solution found (after multiple trials) is used as a comparison. This is referred to as Z^*. All variants of the L2P hyper-heuristics will be compared based on the evolution of the objective function value (w.r.t. Z* in %) as a function of the number of iterations and the execution time (in minutes), in the form of confidence intervals for the estimated P10, P50, and P90 quantiles.

As illustrated in Figure 1, L2P–A2C is by far the best performing variant of the L2P hyper-heuristic, reducing the execution time by 40–75%. L2P—PPO reduces the execution time by 40–65%. Compared to the other variants, L2P–PPO



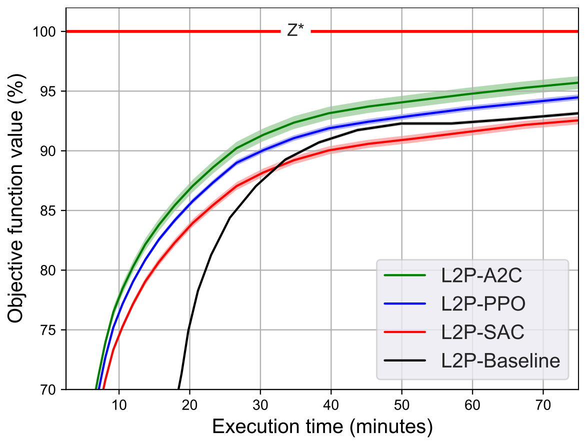


Figure 3: Evolution of the objective function value (in %) for the Instance I_4 (left) and I_5 (right) with respect to the execution time (in minutes).

has the smallest standard deviation in terms of the number of iterations and the execution time. L2P–SAC is the least performing variant, showing that SAC is not a good fit for the current environment. This may be explained by two reasons. First, SAC uses entropy regularization to prevent premature convergence. This can make SAC take much longer to learn compared to A2C and PPO. Second, the trade-off between exploration and exploitation within SAC is controlled by the coefficient α_H, which needs to be changed from one environment to another and requires careful tuning. This makes SAC difficult to include in a ready-to-use solver with little to no pre-tuning and pre-training.

In the third stage of testing, the agents that have already been trained on I_4 are used to solve I_5. As in Figure 3, L2P–A2C outperforms all other variants, reducing the execution time by 45–50%, when compared to L2P–Baseline. L2P–PPO reduces the execution time only by 15–20%. Note that during the optimization, L2P–Baseline outperforms L2P–SAC by providing a better solution. This can be explained by the fact that SAC's policy is not optimal and differentiates minimally between the states during optimization. Thus, the use of L2P–SAC in a real-life scenario seems impractical.

Discussion and Future Research

Reinforcement learning has been shown to adapt the search better and guide it towards better solutions. This opens multiple directions for future research. First, because the proposed score function produced by the RL agent is more adapted both to the current state and future states, updating the score function does not need to be done at each iteration. Thus, the solution can be evaluated only after a pre-fixed or dynamic number of iterations. This is similar to generating multiple columns instead of generating only one column at a time in a column generation algorithm. Second, the input of the RL agent can be further expanded and augmented, and more state-of-the-art agents can be integrated into the hyper-heuristic framework to study further how to better predict the heuristics' future performance based on past performance. Third, a particular architecture can be used within the RL agent to harness the spatiotemporal architecture of the input. Fourth, a multi-agent framework can be developed using a synchronized replay buffer to accelerate the learning phase of the RL agents.

Since the SSOMC is similar to other scheduling problems, exploring more efficient methods for solving such problems becomes even more justified. Indeed, unlike problem-specific heuristics, the presented solution method can be generalized to other problems as in airlines (airline crew scheduling), bus (shift scheduling), trucking (routing problems), and rail industries. The only requirement would be to implement a set of low-level heuristics, none of which would need to be well-tuned. The hyper-heuristic will guide the search towards a better solution while sampling towards the most promising heuristics.

Conclusion

This paper proposes a solution method to solve the SSOMC problem that is of particular interest to the mining industry. The proposed learn-to-perturb (L2P) hyper-heuristic incorporates various state-of-the-art RL models to guide the search towards better solutions and results show its effectiveness on multiple case studies, reducing the computational time by 30-45%. Decision-makers can take advantage of such hyper-heuristics harnessing reinforcement learning to guide the search for a better solution when solving large-scale optimization problems since the proposed approach is not limited to mine scheduling.

Acknowledgments

The work in this paper was funded by the National Sciences and Engineering Research Council (NSERC) of Canada CRD Grant 500414-16, the NSERC Discovery Grant 239019, and the industry consortium members of COSMO Stochastic Mine Planning Laboratory (AngloGold Ashanti, BHP, De Beers/AngloAmerican, IAMGOLD Corporation, Kinross Gold, Newmont Mining Corporation, and Vale), and the Canada Research Chairs Program.

References

Bengio, Y.; Lodi, A.; and Prouvost, A. 2020. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*.

Chatterjee, S.; and Dimitrakopoulos, R. 2019. Production scheduling under uncertainty of an open-pit mine using Lagrangian relaxation and branch-and-cut algorithm. *International Journal of Mining, Reclamation and Environment*, 34(5): 343–361.

Chmiela, A.; Khalil, E. B.; Gleixner, A.; Lodi, A.; and Pokutta, S. 2021. Learning to Schedule Heuristics in Branch-and-Bound. *arXiv preprint arXiv:2103.10294*.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, 15554–15566.

Glover, F.; and Laguna, M. 1998. Tabu search. In *Handbook of combinatorial optimization*, 2093–2229. Springer.

Glover, F. W.; and Kochenberger, G. A. 2006. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

Godoy, M. 2003. *The effective management of geological risk in long-term production scheduling of open pit mines*. Ph.D. thesis, The University of Queensland.

Goodfellow, R.; and Dimitrakopoulos, R. 2017. Simultaneous stochastic optimization of mining complexes and mineral value chains. *Mathematical Geosciences*, 49(3): 341–360.

Goodfellow, R.; and Dimitrakopoulos, R. 2019. COSMO Suite: A platform for optimizing mining complexes with uncertainty. *Les Cahiers du GERAD*, G–2016(96).

Goodfellow, R. C.; and Dimitrakopoulos, R. 2016. Global optimization of open pit mining complexes with uncertainty. *Applied Soft Computing*, 40: 292–304.

Gupta, P.; Gasse, M.; Khalil, E. B.; Kumar, M. P.; Lodi, A.; and Bengio, Y. 2020. Hybrid models for learning to branch. *arXiv preprint arXiv:2006.15212*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 1861–1870. PMLR.

IBM. 2020. CPLEX.

Khan, A.; and Niemann-Delius, C. 2015. Application of particle swarm optimization to the open pit mine scheduling problem. In *Proceedings of the 12th International Symposium Continuous Surface Mining-Aachen 2014*, 195–212. Springer.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *science*, 220(4598): 671–680.

Kumral, M. 2013. Optimizing ore–waste discrimination and block sequencing through simulated annealing. *Applied Soft Computing*, 13(8): 3737–3744.

Lamghari, A.; and Dimitrakopoulos, R. 2016. 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.; and Senécal, R. 2021. A matheuristic approach for optimizing mineral value chains under uncertainty. *Optimization and Engineering*, 1–26.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; and Teller, E. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6): 1087–1092.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PMLR.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Talbi, E.-G. 2009. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

Yaakoubi, Y. 2019. *Combiner intelligence artificielle et programmation mathématique pour la planification des horaires des équipages en transport aérien*. Ph.D. thesis, Polytechnique Montréal.

Yaakoubi, Y.; Soumis, F.; and Lacoste-Julien, S. 2019. Flight-connection prediction for airline crew scheduling to construct initial clusters for OR optimizer. *Les Cahiers du GERAD*, G–2019(26).

Yaakoubi, Y.; Soumis, F.; and Lacoste-Julien, S. 2020. Machine Learning in Airline Crew Pairing to Construct Initial Clusters for Dynamic Constraint Aggregation. *EURO Journal on Transportation and Logistics*.

Yaakoubi, Y.; Soumis, F.; and Lacoste-Julien, S. 2021. Structured Convolutional Kernel Networks for Airline Crew Scheduling. *ICML*.

Zarpellon, G.; Jo, J.; Lodi, A.; and Bengio, Y. 2020. Parameterizing branch-and-bound search trees to learn branching policies. *arXiv preprint arXiv:2002.05120*, 12.