Shovel Allocation and Short-Term Production Planning in Open-Pit Mines Using Deep Reinforcement Learning

Roberto Noriega and Yashar Pourrahimian[1]
Mining Optimization Laboratory (MOL)
University of Alberta, Edmonton, Canada

ABSTRACT

The open-pit production system is a highly dynamic and uncertain environment with complex interactions between haulage and loading equipment on a shared road network. One of the key decisions in open-pit short-term planning is the allocation sequence of shovels to mining faces to meet the production targets established by the long-term strategic plan. Deep Reinforcement Learning (DRL) techniques have been widely applied to dynamic production environments where an agent is trained on a simulation of the production system to learn the best decisions to take given the system's state at any given time. This paper proposes a DRL approach based on the Deep Q-Learning algorithm to obtain a robust shovel allocation plan for open-pit short-term planning. A discrete-event simulation of the mining production system incorporating trucks, shovels, crushers, dumps and the road network is developed, where each component of the equipment operating cycles is subject to uncertainties modelled based on historical activity records to serve as the environment to train the DRL agent. The goal is to learn a robust shovel allocation strategy for the next 3-months to meet the tonnes per hour (TPH) production target to be delivered to the crusher feeds by interacting multiple times with the production simulator. As a result, the agent successfully learns a shovel allocation plan that achieves the goal considering all the operating uncertainties for the case study.

1. Introduction and Background

The open-pit mining production system is a highly dynamic environment that comprises the operation and coordination of multiple pieces of equipment of different types and capacities to achieve a production goal, usually delivering a certain amount of ore within a certain quality range to processing facilities to comply with the long-term plan (Newman et al. (2010) [9]). A major challenge in open-pit short-term planning is the high uncertainty arising from the dynamic interaction of different machines, operating cycle times and failures, and geological uncertainties in the quality of the material being mined. This often leads to hard-to-reach plans at the operational stage due to mismatches in productivity and geological forecasts, which then require frequent efforts to update plans and resolve issues as they appear.

Commercial tools and academic research in open-pit short-term planning focus on developing mathematical programming frameworks, usually linear optimization models or similar heuristics, which require the formulation of large and complex models to capture the highly dynamic open-pit production environment (Blom et al. (2018) [1]). However, a major drawback of these approaches is the complexity in including operational uncertainties, which make an already intractable mathematical problem substantially more complex, with limited capabilities to consider a significant number of production scenarios (Both and Dimitrakopoulos (2018) [2]).

ISBN: 978-1-55195-486-8

Simulation models have been used extensively in mining to estimate the productivity of mining systems by using historical data to reproduce equipment behaviour and interactions to forecast future performance (Raj et al. (2009) [12]). In addition, simulation models provide an efficient approach to quantifying the different operational uncertainties and particularities of the day-to-day operations in open-pit mines. Therefore, researchers have proposed using simulation models to serve as a platform for an optimization engine that provides robust and optimal short-term mine planning decisions (Upadhyay and Askari Nasab (2018) [17]; Shishvan and Benndorf (2019) [14]). However, current simulation-optimization efforts still use linear optimization techniques to find the planning decisions, which struggle to efficiently model the dynamic environment of day-to-day mining operations leading to suboptimal decisions, inability to account for a wide range of production scenarios and large computation times which could render real-world use unfeasible.

This research proposes a Deep Reinforcement Learning (DRL) approach for robust and adaptive open-pit short-term planning, specifically for dynamic shovel allocation and mining sequencing decisions. Reinforcement Learning (RL) is a branch of Machine Learning (ML) that involves a computational approach to learning from interactions with an environment to maximize a goal (Sutton and Barton (2018) [16]). DRL has seen an increased application for optimizing different engineering systems, such as in the transportation, manufacturing and heavy industries, providing a highly flexible data-driven production control framework (Panzer and Bender (2021) [11]).

In an RL framework, an agent, an abstraction for the decision-maker, interacts with an environment at different time steps. At any time step t where the agent must act, it observes the current state of the system, s_t, and makes an action based on it. The environment then responds to this action by transitioning into a new state in the next time step s_{t+1}, and providing a reward R_{t+1} for the agent. This sequential decision-making behavior (Figure 3) repeats itself until the environment transitions into a final state, and the interaction ends; alternatively, the agent could interact with the environment indefinitely, depending on the application.

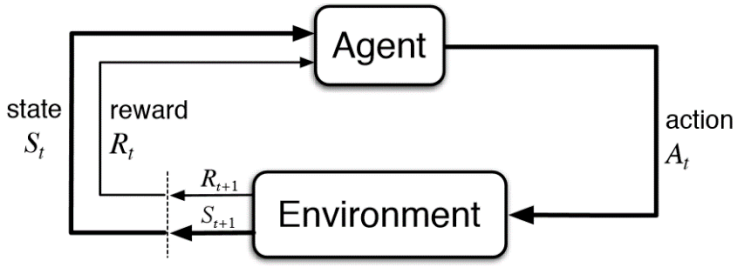


Figure 1. Reinforcement Learning conceptual framework [16].

RL aims to enable the agent to learn an optimal decision-making policy that maximizes the cumulative reward received throughout its interaction. The objective function that RL algorithms optimize is the total discounted reward accumulated by the agent by interacting with the environment. Therefore, the rewards returned by the environment are designed to reflect the desired goals achieved in each application. In the context of open-pit short-term planning, this could profit from ore deliveries to the crusher and penalties from deviations in production targets. The decision-making policy is expressed as mapping, or function, from states of the system to actions to make, and it can be implemented as a hardcoded table of state variables or a complex function approximator as an Artificial Neural Network (ANN). The development of this mapping from states to actions is achieved by a learn by doing approach, where the agent interacts with the environment, exploring it, discovering the impact of unknown actions, and exploiting well-proven high reward decisions. Due to a large number of environment interactions needed to converge to an optimal policy, the agent and environments are commonly implemented as computer simulations before moving to real-world trials and applications (Naeem et al. (2020) [8]).

A practical in the heavy industries for production planning in a chemical plant, more akin to the environment of the extractive industries, is presented by Hubbs et al. [5]. The authors proposed an actor-critic algorithm, policy-based RL, for optimal and robust chemical production scheduling under uncertain demand and equipment operating cycles. The DRL approach was built using historical records and was exhaustively benchmarked against current practices and MIP-based scheduling algorithms, reporting that the DRL scheduling led to increased profitability and better response to unforeseen situations. Another major advantage of DRL scheduling is the fast-computing times in deployment after training since it only requires a forward pass through Deep Neural Network (DNN). This was demonstrated by Wu et al. [19] where a DRL framework was developed to tackle the production planning of medical masks during the COVID-19 emergency, which caused the arrival of a large number of unexpected orders to manufacturing facilities. The DRL was trained to minimize total tardiness in order completion and was tested with data from a medical mask manufacturer, showing that the DRL system could generate production plans and efficiently handle real-time rescheduling of orders significantly better than existing heuristics during the peak of the emergency period.

The literature on DRL applications to production systems across different industries is vast. The readers are directed to Panzer and Bender [11] for a comprehensive review. The authors found that 89% of the benchmarked DRL implementations resulted in improved scheduling performance achieving lower total tardiness, higher profits, or other specific objectives, compared to current practice and other heuristics.

2. Methodology

2.1. Problem Description

Once the strategic plan for a mine is established, the operational and short-term plan requires determining an optimal and feasible sequence of mining areas to be prepared and extracted along with allocating equipment resources for these activities over shorter periods. Commonly operational plans are defined quarterly or month to month for activities such as mining sequences, mine access development and shovel allocations. At this stage, one of the critical decisions is to define a shovel allocation policy that assigns shovels to mining faces to meet ore production and quality targets. This decision is subject to high operational uncertainties in the estimated production outputs due to the stochastic nature of shovel loading and truck haulage operations, and geological and other uncertainties in estimating the rock properties.

For this purpose, many algorithms have been proposed in the literature to solve the short-term planning problem considering different decisions and constraints. Most commonly, Mixed-Integer Linear Programming (MILP) models have been developed to solve the operational short-term planning problem. However, deterministic MILP models do not allow to account for any source of uncertainty which can render their solutions unfeasible, requiring effort in the field to accommodate changes over the initial plan. Moreover, they require describing the model as a set of linear equations however the production environment of open-pit mining is a highly dynamic, uncertain environment which greatly complicates solving MILP models.

In this paper, an Artificial Intelligence (AI) agent is developed to learn a shovel-allocation policy to maintain the production targets at crusher feeds during a production quarter. The AI agent is a Neural Network that given the available mining areas to mine at any point and the shovel that requires an allocation, will suggest a matching that will meet the required production targets. For this purpose, the AI agent is trained in a simulation model of the mine production environment, that is built using historical equipment records to mimic the real mine performance, under Deep Q-Learning, a RL framework. As a result, the mine production environment reflects all the operational uncertainties, and the AI agent learns a shovel allocation policy that directly accounts for them.

2.2. Deep Q-Learning

Q-Learning is one of the most widely used RL based algorithms and has seen success in industrial production scheduling applications (Panzer and Bender (2021) [11]). Q-Learning is based on learning from a trial-and-error approach where an agent interacts with an environment over time steps $t = 1, 2, ..., T$ until a terminal time step T is reached. At every time step, the agent observes the environment state s_t and takes an action a_t, from a set of available actions at that time, after which the environment responds to this action by transitioning to a new state s_{t+1} and providing the agent a reward r_{t+1}. The goal of the agent is to find an optimal policy, action selection strategy, that maximizes the total return at any time step t, G_t, defined as the discounted cumulative reward obtained from that time step until the end of the interaction $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$, where the discount factor γ determines how much the agent cares about long-term rewards relative to immediate gains.

During the training process of a Q-learning agent, it tries to build an estimate of the expected return obtained from taking action from a given state, defined as the action-value function $Q(s, a) = E[G_t | S_t = s, A_t = a]$. For this, the agent explores and exploits its current environment knowledge at every iteration. Exploration refers to selecting an action at random, and exploitation to selecting the action that maximizes the returns given the current knowledge of the environment. The most common strategy to balance the exploration versus exploitation problem is the epsilon-greedy exploration strategy, where at every time step with ε probability the agent explores, and ε is decreased over time.

Once the Q-value function is estimated, the optimal policy is that which maximizes $argmax_{a_t} Q(s_t, a_t)$ at every time step. This would require visiting every state-action pair for a given environment multiple times, which would be impossible for any real-world application. To address this issue, the action-value function is parametrized as a function with some parameters θ, $Q(s, a; \theta)$, that given a state and action vectors predicts the return from the environment. A Neural Network (NN) can be used as the function approximator and is trained to predict the action-value function $Q(s, a; \theta)$ for any state and action pair from interaction with the environment. The implementation details for the use of a NN within a Q-learning framework, Deep Q-Learning (DQL), are fully described in Mnih et. al. (2015) [7].

The training process in DQL relies on the agent interacting with the environment storing experience vectors, $e_t = (s_t, a_t, r_t, s_{t+1})$ that represent each transition observed in a memory replay buffer which serves as the training dataset for every training update of the NN. At every NN training step, a batch of experiences are drawn from the replay buffer, and the NN weights are trained to minimize the prediction loss $L_i(\theta_i)$ defined as the mean square error (MSE) between the observed return (target) and the predicted return from the network at training step i:

$$L_i\left(\theta_i\right) = \left(r + \gamma max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta_i^{tgt}\right) - Q(s_t, a_t; \theta_i)\right)^2$$

Where θ_i^{tgt} refers to a NN used to evaluate target returns not trained at every step but synced with the online network defined by θ_i every C steps, which helps stabilize the training process.

The general algorithm for the original Deep Q-Learning is described below.

Algorithm General DQL framework

Initialize replay memory D to an initial capacity N. Initialize action-value function Q with weights θ and target action-value function Q^{tgt} with weights $\theta^{tgt} = \theta$.

For each episode:

For $t = 1, ..., T$:

Observe environment state s_t

With probability ε select a random action a_t otherwise $a_t = argmax_a Q(s, a)$

Execute action a_t in environment. Observe reward r and next state s_{t+1}. Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in memory replay buffer D

Sample a random batch of transitions from the replay buffer D.

For every transition in the batch, calculate target $y = r$ if episode ended at this step or $y = r + \gamma max_{a_t} Q^{tgt}(s_{t+1}, a_{t+1})$ otherwise

For every transition calculate loss $L = (y - Q(s_t, a_t))^2$

Update θ to minimize loss with respect to model parameters

Every C steps set $Q^{tgt} = Q$

2.2.1. DQL Implementation

Since the publication of the original DQN method [7], many improvements have been proposed to enhance learning efficiency, significantly improving convergence, training stability and sample efficiency. Google DeepMind collected some of the most important improvements to the original DQN and combined them into the Rainbow DQN agent, showing a significant increase in overall performance (Hessel et al. (2018) [4]).

The DQN implementation in this paper includes the following additional components of the Rainbow DQN agent. Note that the general training framework follows the same algorithm described in Section 2.2.

- n-Step DQN

Improves convergence speed and stability by unrolling the action-value function $Q(s, a)$. The original DQN accumulates single-step rewards at each transition; however, using forward-view multi-step rewards as targets often leads to faster learning, as described by Sutton (1988) [15]. The implementation is straightforward: once an action has been taken from a given state, the return (discounted cumulative reward) observed after n steps from that action is used as the target for the action-value prediction. Values of $n = 2$ to $n = 5$ usually yield good learning behavior. In this research $n = 4$ was used.

- Double DQN

The original DQN tends to overestimate action values, leading to training instabilities and convergence problems. This is due to the maximization step in estimating the returns that serves as the target for training. Double Q-Learning was proposed by van Hasselt et al. (2016) [18] as a solution to the maximization bias, where at every time step choosing actions is done from the Q-network $Q(s, a)$, but the target network Q^{tgt} is used to evaluate the target for the updates.

- Noisy networks for exploration

The epsilon-greedy strategy for exploration can be limiting in complex environments. Fortunato et al. (2017) [3] proposed a simple but improved exploration strategy by adding noise to the weights of the NN agent rather than relying on the epsilon-greedy strategy. The noise in the NN model leads to some randomness in the agent's action selection but is adjusted automatically as an additional parameter by backpropagation during training. As training progresses, the NN can learn to ignore the noisy paths through the network at different rates in different parts of the state space, allowing for a form of state-conditional exploration.

- Prioritized experience replay buffer

The original DQN samples experiences from the replay buffer uniformly for every training step; every transition has the same probability of being used in a training step. Schaul et al. (2016) [13] argued that it would be ideal to sample transitions from which there is more to learn more frequently and proposed a prioritized replay buffer mechanism. Transitions are sampled with a probability proportional to that transition's last observed training loss. Therefore, the agent trains more often on transitions from which it had trouble predicting their outcome.

- Dueling DQN

Wang et al. (2016) developed a novel NN architecture suited for value-based RL methods that features two streams of computation, based on the observation that the action-value function $Q(s, a)$ can be decomposed as the sum between the value of the state s, $V(s)$, and the advantage of taking action a from state s, $A(s,\ a)$. The advantage of action can be interpreted as how much extra reward some particular action from a given state yields. The dueling DQN architecture takes the feature vector and processes it through two independent paths: one for predicting the state's value and another for predicting each action's advantage. After that, the values can be summed to obtain the Q-function. This architecture resulted in better training stability and faster convergence.

The integrated agent for shovel allocations to meet production targets in open-pit mining developed in this paper follows the basic DQN algorithm incorporating all the improvements discussed above. The loss function to train the NN used is the MSE, as described in Section 2.2 and the optimizer used in training is ADAM, which has become a reliable NN optimizer that typically requires little tuning [6]. This loss function and optimizer combination has shown to perform well for Rainbow-based DQN agents in small and complex environments (Obando-Ceron and Castro (2021) [10]). The entire framework is implemented in Python's Pytorch deep learning package.

2.3. The Agent

The agent is a fully connected NN with three layers and 128 neurons each. Each layer has a Rectified Linear Unit (ReLU) activation function, which helps to speed up gradient calculation times and control vanishing/exploding gradient problems. Moreover, at every step, the gradient norms are clipped to a norm within 10 to stabilize training further, a common practice described in Zhang et al. (2020) [20]; this means that rare extreme experiences will not cause extreme shifts in the NN parameters.

Since the Dueling DQN architecture is used, there will be two network paths: one for predicting state values $V(s)$ and one for predicting the advantage of taking each action from a given state

$A(s, a)$ as described in Section 2.3. The NN agent architecture is depicted in Figure 2. The agent was implemented using the Python's Pytorch package.

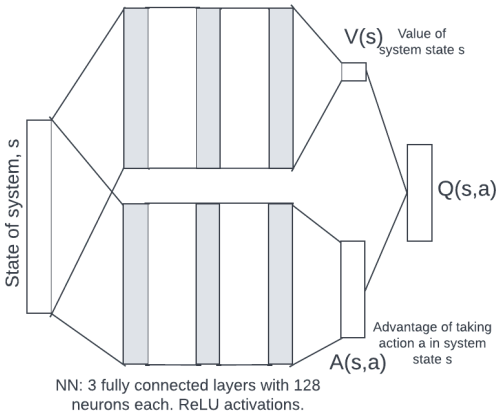


Figure 2. Shovel allocating NN agent architecture.

2.4. The Environment

A discrete-event simulation (DES) model of the operational open-pit truck and shovel environment is developed in Python using the SimPy general-purpose simulation package. The DES models the interaction between loading and hauling equipment, mining faces, crushers, and waste dumps within the mine haul road network and keeps track of different production Key Performance Indicators (KPI) of the system, such as tonnage delivered at crushers, the average grade of ore delivered to crushers amongst other commonly tracked KPIs in mining.

The DES model simulates the extraction of mining faces, aggregation of mineral blocks to be mined by a single shovel, commonly referred to as mining cuts or polygons, and the haulage of material to destination points such as crushers and waste dumps, with the potential to account for stochasticity in every component of the equipment cycles.

The DES starts with an assignment of shovels to their initial mining face and simulates the movement of trucks along the road network to get loaded by the shovels and dump their payload at the set destinations for each mining face. When a mining face is depleted, the shovel needs to be relocated to a new mining face to keep production going; at this point, the AI agent is called to decide which of the available mining faces at that time the shovel will be assigned. Then, the shovel takes some time to move to the new mining area and resumes its operation after arriving. Figure 3 illustrates the general logic.

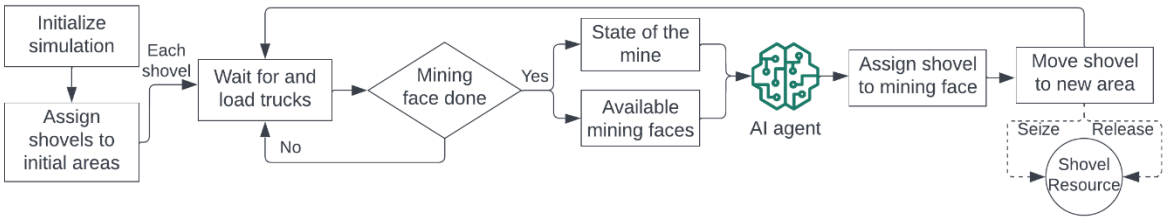


Figure 3. General logic of the open-pit DES model and interaction with the AI agent for shovel allocation decisions.

Truck haulage is modelled by calculating the travel time through the different segments in the road network that form a path between a destination and a shovel, a haul route. The velocities assigned at each individual segment depend on the rimpull curve of the truck and the rolling and grade resistance of the road segment. When a truck arrives at the shovel, it queues and waits for the shovel to be available, then seizes it and receives multiple bucket-loads of material from the mining

face until full. Afterwards, it moves through the haul road network until reaching the destination set for the material coming from the mining face it received its load from, a crusher or a waste dump. If necessary, it queues, dumps its payload, and then travels back to the shovel. Each shovel can break down, which will be unavailable until it is repaired. Figure 4 illustrates the truck haulage logic.

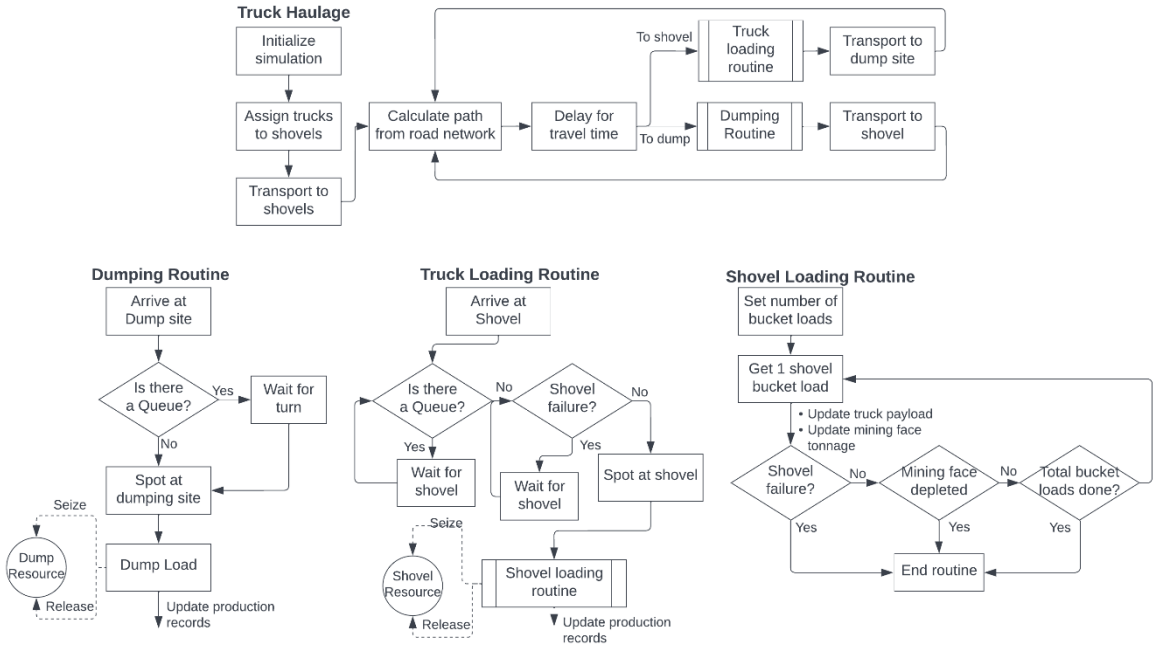


Figure 4. Truck Haulage simulation logic.

Other assumptions made in the current version of the DES model are:

- Each mining face has an average mineral grade and total tonnage. Therefore, each truck payload from a mining face will have the same average grade.
- Destinations for each face's material are fixed; each mining face has a set destination: a given crusher or a given dump. No decisions on destination policies are made at this point but will be considered in future research.
- No truck dispatching logic is considered at this point; each shovel has a fixed truck fleet assigned to it. Future research will consider truck fleet sizes and allow for incorporating a truck dispatching logic.
- No truck bunching through the haul network is considered.

2.5. State and Action Representations

During the training phase, the RL agent learns how to correlate the system state description and the actions taken with the cumulative reward obtained to identify high-value actions. The actions taken by the agent are defined as shovel allocations and happen when a mining face is depleted, and its assigned shovel requires a new mining face allocation to keep production going.

The state of the system at a given time t when an action is required must encode all features needed for the agent to learn its relationship with the desired objective to be maximized. For this purpose, the state of the system is encoded as a vector with the following components:

$$s_t = [MF_i, SH, t]$$

Where MF_i encodes information about every mining face in the system for the time period to be analyzed and is defined as:

$$MF_i = [ton_i, grade_i, dist_i, active_i, avlb_i]$$

Where:

ton_i: Tonnage remaining in mining face i, expressed relative to its total tonnage

$grade_i$: Average mineral grade in mining face i

$dist_i$: Distance from mining face i to its given destination through the road network, normalized between [0, 1] based on the maximum distance across all mining faces

$active_i$: Binary flag: 1 if the mining face is currently being mined by a shovel, 0 otherwise

$avlb_i$: Binary flag: 1 if the mining face is available for mining, 0 otherwise

SH is one hot encoded vector that indicates which shovel needs to be assigned at this time, and t is the current simulation time, expressed as a fraction of the total episode length.

By interacting with the environment, selecting actions using this state representation and observing the total returns (cumulative rewards), the agent learns to predict the value of each action and, based on this prediction, to select an optimal shovel allocation with respect to the reward function.

2.6. Reward to be Optimized

The reward defines the objectives to be maximized. The objective considered here is to minimize the shortage of material delivered to the crusher feed relative to the desired production target $Prod_t$. Therefore, a penalty for production target shortages at the crushers is given to the agent for each step as:

$$Prod_t = -\sum_{h \in H} 1 - \left(\frac{actual\ tph}{target\ tph}, 1\right)$$

Where tph indicates tonnes per hour delivered to the crusher feed, and H indicates the number of hours between the transition. Therefore, the agent is penalized for every hour it fails to meet the target tph between each step, by a value equal to the sum of the relative gaps between the actual tph delivered and the specified target. However, going over the tph target is not penalized, for which the minimum function is used if the actual tph is greater than the target tph.

3. Case Study

The shovel allocation AI agent proposed in this paper was tested on a case study based on an iron ore mining operation. The mining operation uses a total of 5 shovels to load material from mining faces: 2 Hitachi 2500 shovels, with a bucket payload of 12 tonnes, for ore production and 3 Hitachi 5500 Ex shovels, with a bucket payload of 22 tonnes, for waste production. A fleet of 33 trucks is employed to haul the material from the pit to their destination, either one of two crushers or a waste dump. The mine uses 15 CAT785C, with a payload of 140 tonnes, to work with the ore shovels and 18 CAT793C, with a payload of 218 tonnes, to work with the waste shovels. The mine ore

production targets for the crusher feed are 1300 tonnes-per-hour (tph). The mine operates one 12-hour shift per day, seven days a week.

The agent's goal is to define a shovel allocation plan to meet the crusher feed production target for the next quarter (3 months), given the mine layout, equipment performance and available mining faces. The set of mining faces to extract is based on the long-term strategic plan of the mine, where the ones expected to be mined in the next three months are used. Each of these faces has a set of physical precedences that represent the physical space required to start extraction, which is enforced by presenting to the agent only the available faces at each step when an action is required. Figure 5 shows a plan view of the mine layout; in which, in addition to the crusher and waste dump locations, the access to the mining faces areas. From the mining faces access, it is assumed that the distance to each mining face is the linear distance between its digging coordinate and the closest access point in the road network.

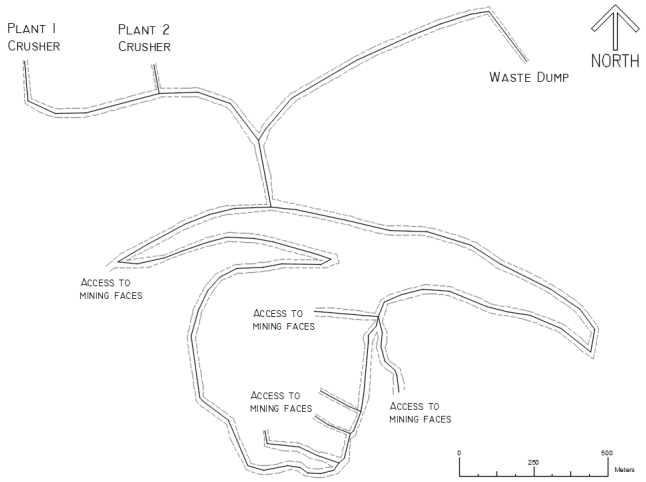


Figure 5. Mine layout for the case study.

To model the equipment production behavior, statistical distributions were fitted to recorded historical data from an available equipment dispatch database to the different activities that comprise the equipment's load and haul operating cycle in the case study. Table 1 shows the equipment distributions. Truck spotting times at the dumping sites were not retrievable, so a practical mean value was used.

Table 1. Distributions fitted to different activities in the productivity cycle of the load and haul equipment.

Activity		Distribution
Shovel bucket cycle time	Hit 2500	Triangular(15, 26, 50) [seconds]
	Hit 5500Ex	Triangular(15, 29, 50) [seconds]
Shovel up-time	Hit 2500	116 * Weibull(34) [hours]
	Hit 5500Ex	116 * Weibull(32) [hours]
Shovel down-time	Hit 2500	Gamma(1.4, 1.5) [hours]
	Hit 5500Ex	Gamma(1.4, 1.5) [hours]
Truck spot time at shovel	CAT 785C	Gamma(22.54, 1.39) [seconds]
	CAT 793C	Gamma(26.91, 1.36) [seconds]
Truck spot time at crusher	CAT 785C	30 [seconds]

	CAT 793C	30 [seconds]
Truck dump time	CAT 785C	Normal(52, 6) [seconds]
	CAT 793C	Normal(55, 8) [seconds]

The truck haulage time throughout the network was determined based on the truck's rimpull characteristics and the road's total resistance. The shortest path between the truck location and its destination is determined, and the travel time is calculated based on the road segments that compose the path by using the maximum speed the truck can achieve on each road segment based on its rimpull curve from the manufacturer specifications and the road total resistance. The mining operation was simulated as described in Section 2.4.

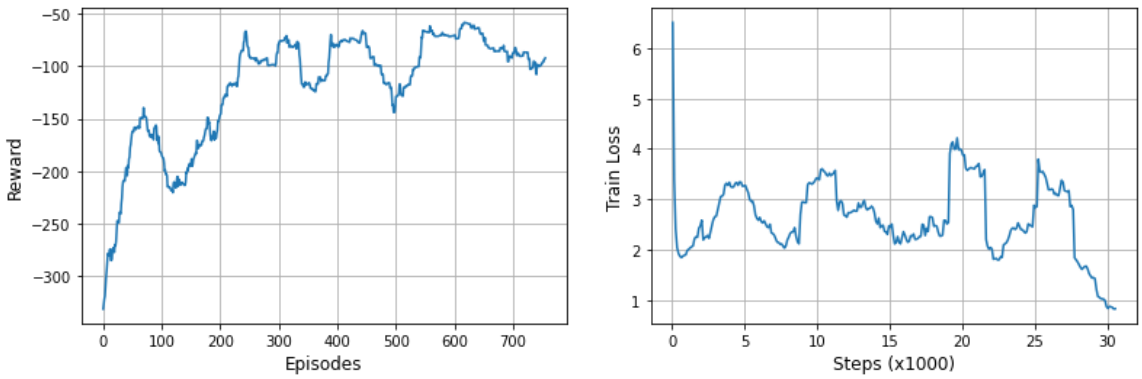
The shovel allocation agent was trained following the DQN algorithm described in Section 2.2, with the hyperparameters shown in Table 2. Future research will investigate each hyperparameter's impact on the agent's training to identify the critical ones and provide some guidelines in selection and tuning.

Table 2. Hyperparameters selected for the training of the AI shovel allocation agent.

Replay buffer size	8000
Initial samples in replay buffer	2000
Batch size for training updates	32
Discount factor	0.99
Learning rate	0.001
Iteration update frequency of target network	1000
n for multi-step returns	4

The training was performed in the Google Colab service, which provides a virtual machine with powerful GPUs to train DL models. The GPU used in the instance where the trained agent was a Tesla P100.

The agent trained for 6 hours until convergence was observed in the reward obtained on each episode, where an episode corresponds to a shovel allocation plan for 3 months (quarter). This indicates that the agent has learned a policy, a decision-making strategy, that achieves the desired goal over multiple potential outcomes based on the stochasticity of the equipment operating cycles and failures, rather than finding a solution to one potential outcome or a completely deterministic scenario based on the average performance. Figure 6 presents the training curves for both the reward achieved by the agent decision-making and the loss from the agent’s NN prediction of shovel allocation action values. Both curves in Figure 6 show a moving average over 25 episodes.



(a) (b)

Figure 6. (a) Reward obtained at each episode during training. (b) Training loss of the agent NN for each training step.

Figure 6 (a) shows how at the start of training, where the agent initializes its NN with random weights, it performs poorly at each episode. The shovel allocation plans at early training stages fail to meet the production targets at the crusher feeds by a large margin, incurring a large negative cumulative reward for the production quarter. However, as training progresses and the agent becomes better at predicting the value of each shovel movement, the performance increases until converging at around -80, with some oscillations due to the stochastic nature of the system. Convergence at this reward level indicates that the agent cannot fully meet the hourly production targets at the crusher feeds due to failure in shovels, which are part of the system and can significantly halt production until repaired.

Figure 6 (b) shows the average loss for the agent's NN prediction at each step during training, where a step means a shovel allocation action and every episode is composed of multiple of these. In the early stages, the NN performs poorly but improves its performance rapidly, providing better predictions of the value of each shovel allocation. Significant oscillations are observed in the loss curve, which are common in DRL applications. Since the NN training data is generated from a decision-making policy that is also changing through the training phase as the agent improves its performance, this means that the distribution of the training examples is changing continually, and the NN is effectively chasing a moving target. The implementation of the target network for evaluation of value functions rather than using the same network that is constantly changing alleviates this issue in practice. The stagnation in the loss curve performance could suggest that the model system state representation maybe insufficient to fully predict the value of each shovel movement, giving the agent additional information such as past shovel allocations, productivity rates, or cycle times could help the agent make better predictions.

To obtain a shovel allocation plan to use in practice, the simulation can be run with a fully trained agent, and the movements can be recorded. Figure 7 shows a Gantt chart feasible shovel allocation plan obtained by the agent for the production quarter. Each horizontal bar represents the allocation of a shovel to a given mining face, based on its ID, for each day. The plan proposed by the agent is robust as after training, the agent found a shove allocation strategy to meet the desired goals over many potential productivity outcomes. Moreover, the agent can be updated with the real-world progress of the operational plan and queried at any time in production that a decision is needed to obtain a suggested shovel allocation action.

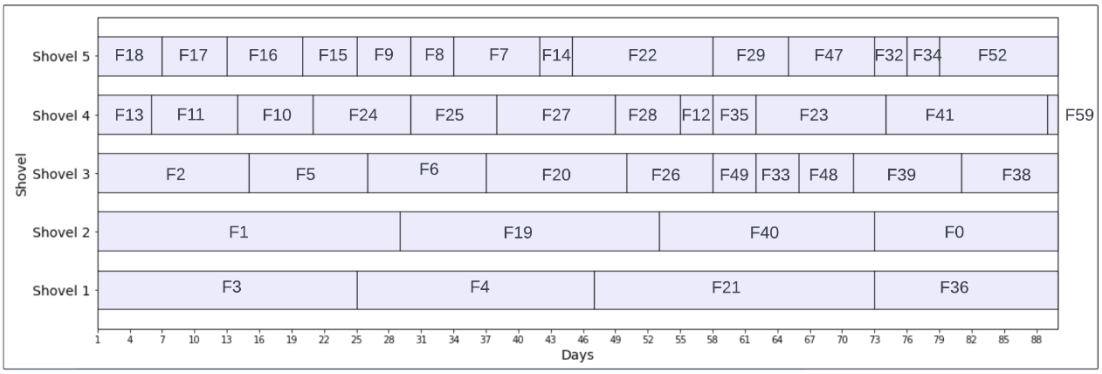
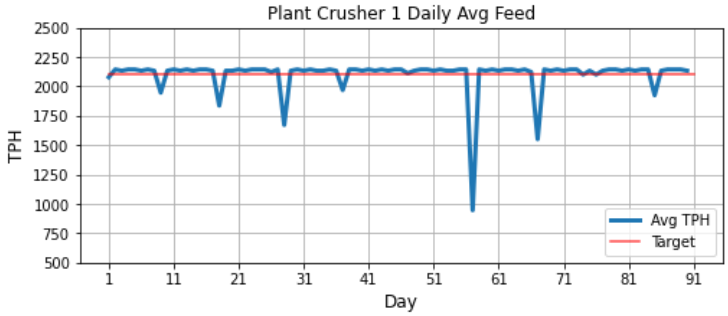
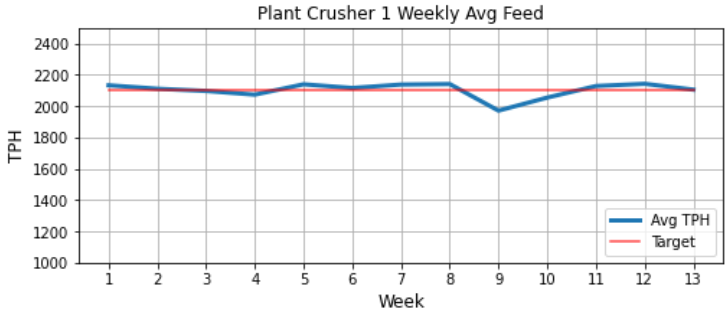


Figure 7. Shovel allocation plan for a production quarter of the case study.

The shovel allocation plan was evaluated by observing the crushers' feed to ensure it meets the desired target. Figure 8 and Figure 9 show the average daily and weekly TPH delivered at the plant crusher 1 feed and plant crusher 2 feed, respectively for the production quarter.



(a)



(b)

Figure 8. TPH delivered to plant 1 crusher feed from the shovel allocation plan. (a) Daily average and (b) weekly average

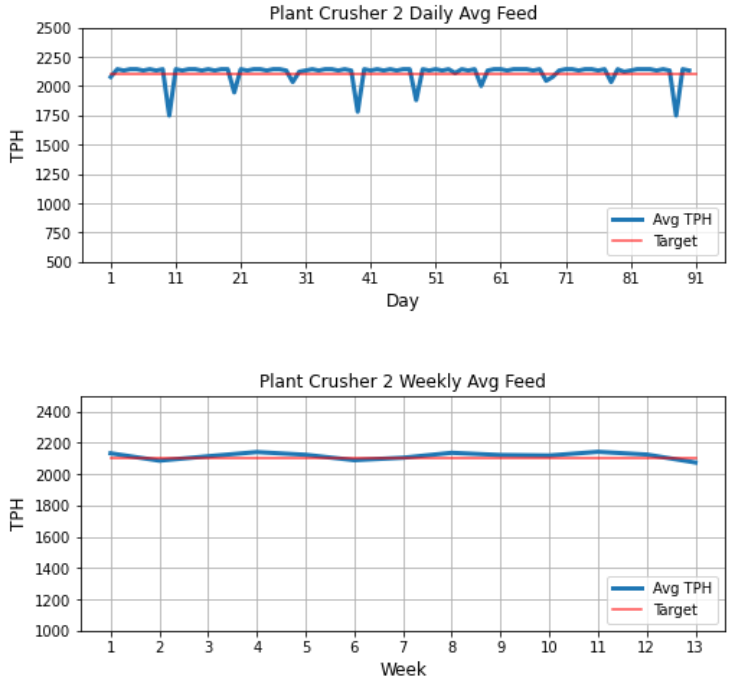


Figure 9. TPH delivered to plant 2 crusher feed from the shovel allocation plan. (a) Daily average and (b) weekly average.

Overall, the agent meets the production goals as closely as possible; the large drops in TPH are due to shovel failures since there are only two ore shovels; the agent cannot feed the crusher they were working on until repaired.

4. Conclusions and Future Research

A simulation-optimization approach for open-pit short-term planning is proposed in this research to obtain a robust shovel allocation plan that meets specified production targets under operational uncertainties of equipment performance. A RL framework is used where a NN based agent allocates shovels throughout the production simulation by observing the state of the mine and the available mining faces. The agent receives a penalty for every hour it fails to meet a specified production target, defined as tonnes per hour (TPH) delivered to crusher feeds, equal to the relative gap from the actual TPH observed in response to the shovel allocation actions. A Deep Q-learning RL framework is used to train the agent to learn an optimal allocation policy to minimize this production shortage over multiple interactions with the mine production simulator. During training, the agent's NN gets better at predicting the return of shovel allocation actions given the state of the mine and available mining faces, where the return is defined as the long-term cumulative reward obtained which gives the agent some insight into the long-term impact of each action, and it's guided towards high-value actions to define an optimal plan.

A case study is presented for an iron ore mine where a shovel allocation plan is required for the next production quarter (3 months). The agent was trained for 6 hours, and its performance converged to a shovel allocation policy that met the specified TPH target delivered at two plant crusher feeds. This plan is robust as the agent has interacted with the environment multiple times, and the strategy learned produces a similar total return over many production simulations.

Future research will be directed into different state representations and NN architectures that can enhance learning efficiency. Currently, a simple approach of representing the system's state as a long vector serving as input to a basic fully connected NN is proposed, which can be improved by

investigating NN architectures more suitable for learning long-term dependencies or graph-structured problems. Moreover, additional feature engineering can also improve the learning efficiency of the agent, by improving the information used in the state representation. More complex rewards will be investigated too, including operating costs and blending to learn a minimum cost feasible short-term plan.

5. References

[1] Blom, M., Pearce, A. R., and Stuckey, P. J. (2018). Short-term planning for open pit mines: a review. *International Journal of Mining, Reclamation and Environment*, *33* (5), 318-339.

[2] Both, C. and Dimitrakopoulos, R. (2020). Joint stochastic short-term production scheduling and fleet management optimization for mining complexes. *Optimization and Engineering*, *21* (4), 1717-1743.

[3] Fortunato, M., Azar, M., Piot, B., Menick, J., Hessel, H., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2018). *Noisy Networks for Exploration*. in Proceedings of 6th International Conference on Learning Representations,

[4] Hessel, H., Modayil, J., van Hasselt, H., Schaul, T., Ostrovoski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). *Rainbow: Combining Improvements in Deep Reinforcement Learning*. in Proceedings of Proceedings of the AAAI Conference on Artificial Intelligence,

[5] Hubbs, C. D., Li, C., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020). A deep reinforcement learning approach for chemical production scheduling. *Computers & Chemical Engineering*, *141* 106982.

[6] Kingma, D. and Lei Ban, J. (2015). *ADAM: A Method for Stochastic Optimization*. in Proceedings of Internatioanl Conference on Learning Representations 2015, San Diego, CA,

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518* (7540), 529-33.

[8] Naeem, M., Tahir, S., Rizvi, H., and Coronato, A. (2020). A Gentle Introduction to Reinforcement Learning and Its Application in Different Fields. *IEEE Access*, *8* (209320),

[9] Newman, A., Rubio, E., Caro, R., Weintraub, A., and Eurek, K. (2010). A review of operations research in mine planning. *Interfaces (Providence)*, *40* 222-45.

[10] Obando-Ceron, J. and Samuel Castro, P. (2021). *Revisiting Rainbow: Promoting more Insightful and Inclusive Deep Reinforcement Learning Research*. in Proceedings of Proceedings of the 38th International Conference on Machine Learning,

[11] Panzer, M. and Bender, B. (2021). Deep reinforcement learning in production systems: a systematic literature review. *International Journal of Production Research*, 1-26.

[12] Raj, M., Vardhan, H., and Y., R. (2009). Production optimisation using simulation models in mines: A critical review. *International Journal of Operational Research*, *6* (3), 330-359.

[13] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). *Prioritized Experience Replay*. in Proceedings of 4th International Conference on Learning Representations,

[14] Shishvan, M. S. and Benndorf, J. (2019). Simulation-based optimization approach for material dispatching in continuous mining systems. *European Journal of Operational Research*, *275* (3), 1108-1125.

[15] Sutton, R. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, *3* 9-44.

[16] Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, 2nd ed,

[17] Upadhyay, S. P. and Askari-Nasab, H. (2018). Simulation and optimization approach for uncertainty-based short-term planning in open pit mines. *International Journal of Mining Science and Technology*, *28* (2), 153-166.

[18] van Hasselt, H., Guez, A., and Silver, D. (2016). *Deep reinforcement learning with double Q-Learning*. in Proceedings of AAAI'16: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094-2100.

[19] Wu, C. X., Liao, M. H., Karatas, M., Chen, S. Y., and Zheng, Y. J. (2020). Real-time neural network scheduling of emergency medical mask production during COVID-19. *Appl Soft Comput*, *97* 106790.

[20] Zhang, J., He, T., Sra, S., and Jadbabaie, A. (2020). *Why gradient clipping accelerates training: A theoretical justification for adaptivity*. in Proceedings of 8th International Conference on Learning Representations,