Les Cahiers du GERAD ISSN: 0711–2440

Mineral supply chain optimization under uncertainty using approximate dynamic programming

C. Paduraru
R. Dimitrakopoulos

G–2015–94

September 2015

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

Mineral supply chain optimization under uncertainty using approximate dynamic programming

Cosmin Paduraru
Roussos Dimitrakopoulos

GERAD & COSMO – Stochastic Mine Planning Laboratory, Department of Mining and Materials Engineering, McGill University, Montreal (Quebec) Canada, H3A 2A7

cosmin.paduraru@mail.mcgill.ca
roussos.dimitrakopoulos@mcgill.ca

September 2015

Les Cahiers du GERAD
G–2015–94

Abstract: The optimization of mine complexes and related value chains is a challenging problem due to the simultaneous presence of a highly-dimensional decision space, integer decision variables, non-linear constraints and recovery functions, as well as geological and market uncertainty. In addition, if the uncertainty is revealed progressively then decisions need to continuously adapt to new information, which poses additional challenges. Stochastic dynamic programming is a well-known class of methods for addressing problems where a decision-maker reacts to new information. While simplistic applications of dynamic programming using exact representations would be intractable for any realistic mine planning problem, new dynamic programming methods using approximate value function representations have been successfully implemented in the past to address problems of similar complexity. This work formulates the problem of mine complex optimization as an approximate dynamic program, and presents an application of the resulting method to a gold-copper deposit.

1 Introduction

In a mining complex, the mineral extracted from the ground can follow different blending and processing streams. These streams, and the material transformations that occur along the way, can be seen as forming a mineral supply chain. Optimizing the quality and quantity of the inputs to the metallurgical processes within the supply chain can increase the value of the mining complex. This processing stream optimization can also lead to a higher quality mine schedule, by allowing the scheduler to compute block values according to the actual maximum-value destination, instead of relying on simplifying assumptions needed to compute processing-stream-independent economic block values.

There has been increasing interest in recent years in the global optimization of mining complexes (Goodfellow and Dimitrakopoulos, 2012, 2013; Montiel and Dimitrakopoulos, 2013; Whittle, 2010, 2007). This is a highly complex problem due to the presence of a large number of integer decision variables, the availability of multiple processing streams, and potential non-linearities introduced by recovery curves or blending requirements. In addition, uncertainty needs to be taken into account by the optimization models in order for realistic solutions to be produced. Indeed, previous work has shown that ignoring geological uncertainty can lead to lower-quality solutions for both stand-alone scheduling problems (Dimitrakopoulos, 2011) and global optimization problems (Goodfellow and Dimitrakopoulos, 2013; Montiel and Dimitrakopoulos, 2013). Another important source of uncertainty is given by potential market fluctuations (Del Castillo and Dimitrakopoulos, 2013; Johnson et al., 2011).

Adding even further to this complexity, mining operations may have the flexibility to adapt their operation according to new information obtained. For instance, managers may have an option to close the mine earlier or later than originally planned (Del Castillo and Dimitrakopoulos, 2013; Dimitrakopoulos and Abdel Sabour, 2007), or to adapt their production rates in order to respond to price changes (Kizilkale and Dimitrakopoulos, 2013). Accounting for this flexibility can better prepare management by foreseeing how they will respond to uncertain future information. Similarly to processing stream optimization, this can provide more accurate material values for use by scheduling methods.

Previous work dealing with flexibility modeled either CAPEX-related decisions, such as life-of-mine or production rate, or scheduling-related decisions (Boland et al., 2008). This paper introduces a new method for optimizing processing stream decisions under uncertainty.

The proposed method is based on dynamic programming (Bellman, 1957; Cormen et al., 2001; Puterman, 1994), which efficiently combines solutions to different subproblems of the original optimization problem. In order to deal with the high dimensionality of the problem representation space, we use approximate dynamic programming (Bertsekas and Tsitsiklis, 1996; Powell, 2007; Sutton and Barto, 1998), which works by approximating the value of the subproblems' solutions. Approximate dynamic programming has been used in large-scale industrial applications such as fleet management (Powell et al., 2012a) and power system control (Powell et al., 2012b; Ernst et al., 2009).

There are a few examples in the literature of dynamic programming applied to mine planning. Dowd (1976) proposed a dynamic programming approach for computing a different cut-off grade for each period. A dynamic programming model was also proposed by Wang et al. (2008) for determining per-period cut-off grades as a function of the amount of previously extracted mineral in an underground mine. Kizilkale and Dimitrakopoulos (2013) used dynamic programming and game theory to compute a strategy for adapting mine production rates in response to price changes in a multi-mine complex. Because they used a target-tracking formulation of the problem, they were able to use quadratic control (Anderson and Moore, 1989) and did not require approximations. To our knowledge, this work is the first to use dynamic programming for processing stream optimization, and to use approximate dynamic programming for mine planning in general.

The dynamic programming methodology is introduced in the next section, together with a complete formulation for the problem of computing initial material destination policies. Subsequently, results obtained by applying the proposed method to a copper-gold deposit data set are presented. The paper ends with conclusions and directions for future work.

2 Approximate dynamic programming

Dynamic programming is an approach to solving complex optimization problems by breaking them into smaller subproblems. It works by decomposing the objective function of the original problem into partial evaluation functions corresponding to each of the subproblems, and then combining the subproblems using the partial evaluation functions as a guide.

In order to illustrate how dynamic programming can be used for mineral supply chain optimization, we will present an in-depth example of its application to the problem of allocating material to initial destinations.

2.1 Initial destination policies

This section describes the problem of allocating mined blocks to initial processing destinations. Similarly to Goodfellow and Dimitrakopoulos (2012), it is assumed that the material that needs to be allocated has a set of additive properties. Examples of such properties are total tonnage and metal tonnage, but not metal grade, since this is not an additive property.

Using the concept of additive properties, the output of some initial destination *dest* for a given period t can be computed according to the following general steps:

- Compute the set of blocks *B(dest, t)* that are allocated to *dest* in period t
- Sum up the properties of all blocks $b \in$ *B(dest, t)*; denote the resulting vector by p_{total}*(dest, t)*
- Compute any auxiliary variables based on p_{total}*(dest, t)* that are required in order to compute the output of *dest*; for instance, compute the head grade if a grade-recovery curve is to be used
- Compute the output of *dest* as a function of p_{total}*(dest, t)* and the auxiliary variables computed in the previous step

The revenue obtained from selling the output of the initial destinations is then used as part of the objective function, which has the general form

$$Obj = \sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{dest} E\left[Revenue^{dest}(t) - Cost^{dest}(t) - Penalties^{dest}(t)\right]$$

where T is the total number of periods, and d is a discount factor. Different types of costs (such as processing, selling, or environmental remediation) and penalties (such as deviations from targets or blending constraint violations) can be incorporated.

Let the objective function for a particular scenario *sc* be denoted by Obj_{sc}, such that

$$Obj \approx \frac{1}{\#scenarios} \sum_{sc=1}^{\#scenarios} Obj_{sc} \ .$$

The expectation on the right-hand side is taken with respect to geological and/or financial uncertainty. The expectation can be approximated in a Monte-Carlo fashion by computing the average over a set of scenarios, where each scenario corresponds to a realization of the uncertainty.

The goal of the optimization problem we are interested in is to compute a block destination policy that maximizes the value of Obj. However, the stochastic formulation introduces a new level of complexity to the problem: depending on when the stochasticity is revealed, we can have different types of destination policies, as described in the following sections.

2.1.1 Scenario-independent policies

A scenario-independent policy decides the mapping of blocks to initial destinations before any information about the stochasticity is revealed. This mapping is typically computed so that it is robust with respect to uncertainty, and scenario-independent policies have been shown to be capable of increasing a project's

value while reducing risk (Goodfellow and Dimitrakopoulos, 2013; Montiel and Dimitrakopoulos, 2013). An important advantage of scenario-independent policies is that they have fewer parameters than the other types of policies we will discuss, and consequently they are easier to represent and conceptualize.

We will illustrate the different types of policies discussed in this paper using a very simple example with two blocks, two scenarios, and two destinations. The two destinations, denoted by D1 and D2, have recovery rates of 0.5 and 0.1, respectively. However, D1 has a capacity constraint of 10 tons for the total tonnage of the material sent to it. We assume that the penalty for surpassing the upper bound at D1 by one ton is larger than the revenue that would be obtained from both blocks combined, under either scenario, so that an optimizer should never choose to surpass the capacity of D1.

The block property values under the two scenarios are shown in Table 1. For the purpose of this example, we assume that the revenue obtained is directly proportional to the amount of concentrate produced, and that there are no costs to be considered.

Table 1: Block property values for the example in this section.

b1 properties	**Scenario 1**	**Scenario 2**
Total Tonnage	5	7
Metal Content	0.01	0.02
b2 properties	**Scenario 1**	**Scenario 2**
Total Tonnage	5	4
Metal Content	0.02	0.04

An optimal scenario-independent policy would send b2 to D1 and b1 to D2, because the average profit obtained is higher from sending b2 to D1 than from sending b1 to D1. Note that the policy will never choose to send both blocks to D1 because it risks surpassing the capacity by one ton, if s2 is the actual realized scenario, and thus incurring a prohibitively high penalty.

On the other hand, if the manager had the flexibility to choose the destination for b2 after the property values for b1 have been observed, then both b1 and b2 could be sent to D1 if the tonnage for b1 had been revealed to be equal to 5 tons. However, scenario-independent policies do not take advantage of this type of flexibility. Ignoring the potential to adapt to new information, when such potential exists, can result in overly pessimistic block evaluations, and can therefore reduce the quality of a schedule that is based on these evaluations.

2.1.2 Per-scenario policies

Per-scenario policies provide a separate mapping of blocks to destinations for each scenario. Within each scenario, there is no uncertainty surrounding block properties. Therefore, the allocation decision made for a specific block by a per-scenario policy depends on the exact properties of the other blocks, thus avoiding the potential over-pessimism of scenario-independent policies. Per-scenario policies can be seen as the second part of a two-stage stochastic programming approach to global mine complex optimization (Goodfellow and Dimitrakopoulos, 2012).

For the example described in the previous section, an optimal per-scenario policy would send both blocks to D1 under the first scenario, but the first one to D2 and the second one to D1 under the second scenario. This example illustrates the flexibility of per-scenario policies – in this case, they are able to exploit the fact that, as soon as s1 is identified as actual scenario, both b1 and b2 can be sent to D1 without violating capacity constraints.

On the other hand, the example also illustrates an important drawback of per-scenario policies. While they are able to model the flexibility of adapting the destination according to the observed properties of the material, they cannot model the setting where the properties of some of the un-allocated blocks are not fully known yet. In our example, if the uncertainty for b1 is revealed and it turns out that b1 has a weight of 6 tons,

and therefore is consistent with scenario 2, a per-scenario policy would be computed based on the tonnage for b2 that is given by scenario 2. This implicitly assumes that the uncertainty related to b2 is revealed as soon as the uncertainty for b1 is revealed. However, the properties of the blocks that need to be allocated may not become all known at the same time. If this is the case, then per-scenario policies may produce overly optimistic block evaluations. The next section introduces a class of policies that can model the flexibility to adapt to new information without requiring that all this information be revealed simultaneously.

2.1.3 Per-state policies

We propose per-state policies as a bridge between scenario-independent and per-scenario policies. Per-state policies take their name from the concept of system state, which can be informally described as a numerical representation containing all the necessary information about the system that is being optimized. For a more in-depth and theoretical discussion of the concept of state, see Chapter 5 of Powell (2007).

The concept of state can be illustrated using the example of initial destination policies. In this case, the system to be optimized is composed of the material that needs to be allocated and the initial destinations. Assume that allocation decisions are made in sequence, as described in the previous section. One example of a suitable state representation would be one that includes a numerical summary of the distribution of property values for the blocks that have not yet been allocated, and the aggregate properties of the material already sent to each of the initial destinations. However, making decisions based on distributional representations is a significant challenge, and is beyond the scope of this work. Therefore, the rest of this paper will use a state representation that only contains the aggregate properties of the material sent to the different destinations.

As the name suggests, the decisions made under a per-state policy are a function of the state. Because the state summarizes the change brought to the system by the new information, per-state policies allow us to express the idea of a system that reacts to new information. In the previous example, the property values for b1 may be revealed while the property values for b2 are still uncertain. An optimal per-state policy would send b1 to D1 if its tonnage is equal to 5, but to D2 if it is equal to 7. Note that this is the optimal decision even if the scenario is not fully identified (in other words, even if the values for b2 are still uncertain).

Per-state policies are not necessarily the only way of modelling the process of adapting the policy in response to new information. Multistage stochastic programming (MSP) (Birge and Louveaux, 1997; Boland et al., 2008) can be seen as a way of constraining per-scenario policies in order to ensure that they are not overly optimistic. This is performed through the addition of non-anticipativity constraints. These require that, at every stage in the decision-making process, all scenarios sharing the same previous realizations make the same decision. The relationship between MSP and state-based dynamic programming has been analyzed by various researchers (Powell, 2012; Powell et al., 2012a; Defourny et al., 2012). This work uses per-state policies rather than MSP; this is in large part due to the fact that per-state policies allow for machine-learning-based approximation schemes to be used in order to reduce problem complexity.

2.2 Computing destination policies using dynamic programming

This section illustrates how per-state policies can be computed using dynamic programming, using once again the problem of optimizing initial material destinations as an example. For simplicity of exposition, we focus on optimizing the destination policy within a single period. Extensions to the multi-period setting are possible, but not presented here, since they are not necessary for the case study presented in this paper. This is because in the case study stockpiling is not used and a pre-determined mining schedule is provided, making it possible to treat each period independently.

In order to illustrate the dynamic programming methodology, the concept of state updates must first be discussed. For the initial destination problem, the first state corresponds to the setting where no material has been allocated yet, so the aggregate material properties at the destinations are equal to zero. Every time a new block is allocated, the state is updated to reflect this new allocation. This is done by adding the property values for the block that is allocated to the aggregate properties of the destination it is allocated to.

For instance, for the earlier example with two blocks and two destinations, the first state is

$$s_1 = (D1_{tonnage} = 0,\ D1_{metal} = 0,\ D2_{tonnage} = 0,\ D2_{metal} = 0)\,.$$

Assume that the observed properties of b1 are $(5,\ 0.01)$, in agreement with scenario 1, and that the policy sends b1 to destination D1. Denote by f the function that takes a state and a destination as an input, and produces the updated state as an output. Then we can write

$$f\,(s_1, D1) = (D1_{tonnage} = 5,\ D1_{metal} = 0.01,\ D2_{tonnage} = 0,\ D2_{metal} = 0)\,.$$

The next step is to break down the objective function into terms corresponding to each state update. Denote the first state (before any blocks are allocated) under scenario sc by s_1, and the state after blocks $b_1, \ldots b_{k-1}$ have been allocated according to policy π by s_k^π for all $k > 1$. Note that the state value depends on the scenario, however we will not make this notation explicit in order to avoid overburdening the formulas. Define Obj_{sc} to be the value of Obj_{sc} that would be obtained if all the blocks were allocated according to π in scenario sc. Also define $Obj_{sc}(s)$ to be the value of Obj_{sc} that would be computed assuming that the properties of the entire material sent to the destinations in the current period are given by state s. Note that, under these definitions, $Obj_{sc}^\pi = Obj_{sc}(s_n^\pi)$. Then the value of Obj_{sc}^π can be written as

$$Obj_{sc}^\pi = Obj_{sc}\,(s_n^\pi) = Obj_{sc}\,(s_2^\pi) - Obj_{sc}\,(s_1) + Obj_{sc}\,(s_3^\pi) - Obj_{sc}\,(s_2^\pi) + \cdots + Obj_{sc}\,(s_n^\pi) - Obj_{sc}\left(s_{n-1}^\pi\right).$$

The equation above assumes that $Obj_{sc}\,(s_1)$ is equal to zero, because s_1 corresponds to the situation where no blocks have been allocated yet, and therefore there is no material to process.

For some state s_k and scenario sc, define $C_{sc}\,(s, dest) = Obj_{sc}\,(f(s_k, dest)) - Obj_{sc}(s_k)$, the direct contribution to the objective function made by sending block b_k to destination $dest$. Also define the remaining total contribution of the blocks that still need to be allocated after b_k has been sent to its destination to be

$$C_{sc}^{rem}\,(s_{k+1}, \pi) = \sum_{j=k+1}^{n} C_{sc}\,(s_j, \pi\,(s_j))$$

This notation allows us to re-write Obj_{sc}^π as

$$Obj_{sc}^\pi = C_{sc}^{rem}\,(s_1, \pi) = \sum_{j=1}^{n} C_{sc}\,(s_j, \pi\,(s_j))\,,$$

where the states s_j in the equation above obey the property that $s_{j+1} = f\,(s_j, \pi\,(s_j))\,, \forall j$.

We can now apply a well-known dynamic programming result, the policy improvement theorem (Bellman, 1957), to the initial destination problem. The policy improvement theorem states that, for any policy π, a new policy π' can be computed such that $Obj^{\pi'} \geq Obj^\pi$ by making a local modification to π. For our problem, the modification requires that instead of allocating block b_k to the destination given by $\pi(s_k)$ we allocate it to the destination $dest$ that maximizes

$$E\left[C_{sc}\,(s_k, dest) + C_{sc}^{rem}(f\,(s_k, dest)\,, \pi)\right].$$

Recall that we assumed that the uncertainty is being reduced one block at a time. This means that at step k the property values for all blocks up to and including b_k have been observed and are therefore known with full certainty, but the values for the remaining blocks are still uncertain. Therefore, we can re-write the quantity that $dest$ needs to maximize as

$$C\,(s_k, dest) + E\left[C_{rem}^{sc}(f\,(s_k, dest)\,, \pi)\right],$$

where $C\,(s_k, dest)$ is computed deterministically according to the scenario that corresponds to the observed properties of b_k.

The policy improvement theorem also guarantees that, if we keep improving the policy in this fashion until the decisions made by the improved policy are identical to the decisions made by the old policy, the objective function is maximized for this last policy. Therefore, the dynamic programming algorithm for the initial destination problem will repeat policy improvement steps until this termination criterion is reached, or until some maximum number of iterations if computational constraints need to be taken into account. This iterative process requires an initial policy for the first iteration. This policy can be taken to be the "greedy" policy, that is, the policy that only maximizes the local contribution $C(s_k, dest)$ and ignores the expected total remaining contribution.

There remains, however, one difficulty, which is estimating $E[C^{sc}_{rem}(s,\pi)]$ for all appropriate values of s. For any particular value of s, it can be estimated by simply averaging the sum of the values of $C^{sc}_{rem}(s,\pi)$ over all scenarios. However, the difficulty with this direct approach lies in the iterative nature of the policy improvement process. When we estimate the values of $E[C^{sc}_{rem}(s,\pi')]$ for the improved policy π', we need to keep in mind that π' is a function of the values of $E[C^{sc}_{rem}(s,\pi)]$ corresponding to the old policy π. Therefore, in order to estimate the value of $E[C^{sc}_{rem}(s,\pi')]$ at step k, $(n-k)*(\#scenarios)$ values of $E[C^{sc}_{rem}(s,\pi)]$ need to be estimated. After m iterations of the improvement process, this computational effort grows to $((n-k)*(\#scenarios))^m$ estimations for each allocation. This exponential trend is clearly unsustainable, and we will therefore use the approximation methods outlined in the following section in order to reduce the computational burden.

2.3 Approximating the expected remaining contribution

As discussed in the previous section, computing the expected total remaining contribution can become intractable due to the iterative nature of the policy improvement process. This section shows how the computational burden can be alleviated by using approximation methods based on machine learning, and uses randomized tree ensembles as an example of such a method.

As mentioned before, $E[C^{sc}_{rem}(s,\pi)]$ can be estimated for any policy π by averaging the values of $C^{sc}_{rem}(s,\pi)$ over all scenarios. This is efficient as long as π is not expensive to evaluate. However, when we perform the iterative process this direct estimation method quickly becomes inefficient because π needs to be computed using the estimated values for all the previous policies. In order to avoid this, we will approximate $E[C^{sc}_{rem}(s,\pi)]$ using regression methods in order to be able to evaluate it efficiently.

In general, univariate regression analysis provides an estimate of some conditional expectation $E[Y|x]$, where $x \in \mathbb{R}^n$ and Y is a scalar random variable whose distribution depends on x. Regression analysis requires training data of the form $(x_i,\ y_i)$, where the x_i are called the training inputs and the y_i are called the training outputs or targets. Some algorithmic procedure is then followed in order to estimate $E[Y|x]$ based on these samples. This algorithmic procedure can take a variety of forms depending on factors such as the assumptions on how Y depends on x, the amount of data available, or the desired bias-variance trade-off. Note that regression methods can produce estimates of $E[Y|x]$ for any point x, and not only for the training examples x_i.

In our case, the conditional expectation we want to estimate is $E[C^{sc}_{rem}(s,\pi)]$, which is implicitly conditioned on the value of s. Therefore, the training data set used contains the state values as the x-values and average-based estimates of $E[C^{sc}_{rem}(s,\pi)]$ as y-values.

The particular regression method we use is called extremely randomized trees (Geurts et al., 2006), and comes from the machine learning community. Extremely randomized trees are part of a class of methods that use ensembles of classification or regression trees.

A regression tree (Breiman et al., 1993) is a regression method that uses a tree-based representation to partition the input space into different regions, and predicts a different output value for each of those regions. A regression tree for our problem would partition the input-space, which is the space of state values, according to the different components of the state vector. In our case, these components are the aggregate properties of the material that has already been sent to the initial destinations.

The output returned by a decision tree when a new input is presented is determined by the value stored at the leaf node corresponding to that input. For instance, for the tree in Figure 1, if the input is a state vector for which the total tonnage at D1 is equal to 5 and the metal content at D1 is equal to 0.01, then the returned output V1 is obtained by following the path in the tree corresponding to the input state.

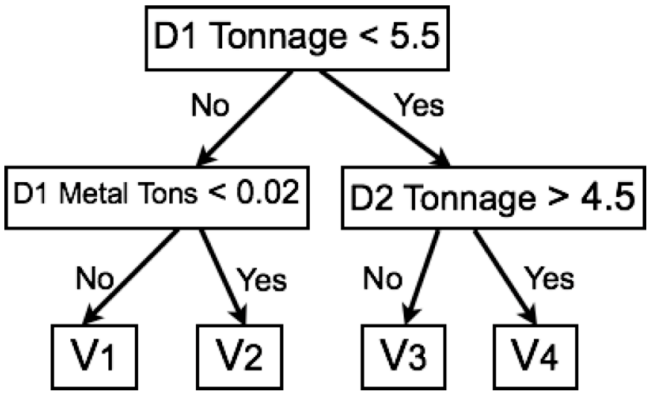

Figure 1: Example of a regression tree.

For a regression tree with a pre-defined structure, the values stored at the leaf nodes are computed based on the training data. That is, the output of a given leaf node is computed as the average of the training outputs for which the respective inputs correspond to a path to that leaf node.

In addition to the leaf node values, the input variables that the tree branches on at each internal leaf (D1 tonnage for the root note of the tree in Figure 1), and the values that those variables are compared to (5 for the root node in Figure 1) need to be computed as well. Regression trees typically compute these values such that the post-split reduction in the entropy of leaf node values is maximized. In other words, they are computed so that the distributions of leaf node values in the resulting subtrees are as peaked as possible.

However, because the value of the split criterion (the reduction in entropy) is estimated using the training data, traditional regression trees can suffer from overfitting. Randomized tree ensembles have been proposed in order to alleviate this issue. A regression tree ensemble is simply a set of regression trees such that, for any input value, the output produced by the ensemble is the average of the outputs produced by the trees in the set for that input. In a randomized ensemble, all the trees are constructed based on the same data set, but randomization is artificially introduced in order to ensure that the trees are different from each other.

There are two main ways to inject randomization that have been proposed in the literature. Random forests (Breiman, 2001) re-sample, with repetition, copies of the original training set, and then construct a separate tree for each re-sampled set using traditional methods. Extremely randomized trees (Geurts et al., 2006), on the other hand, use the original data set without re-sampling for every tree in the ensemble. Instead, they inject randomization by selecting the split characteristics for each internal node (what input variable to branch on, and what value to compare that variable to) randomly. Extremely randomized trees have been shown to have very good empirical performance on both classification and regression tasks, and are the method that is implemented for the case study presented in this paper.

Putting everything together, the approach proposed in this paper proceeds according to the following sequence:

- Start with some initial destination policy π (e.g. greedy)
- Generate training data for regression tree ensemble
  - Generate states s corresponding to different scenarios as inputs
  - Compute average-based estimates of $V_{\pi}^{rem}(s) = E\left[C_{sc}^{rem}\left(f\left(s, dest\right), \pi\right)\right]$, where f is the state update function defined earlier

- Train tree ensemble using the generated training data
- Let π' be a new policy selecting at each state the destination for which $C\left(s_k,\ dest\right) + \widehat{V}_{\pi}^{rem}\left(f(s, dest)\right)$ is maximized, where $\widehat{V}_{\pi}^{rem}$ is the estimate of V_{π}^{rem} computed using the tree ensemble
- Repeat the above steps with $\pi = \pi'$, until the two policies are identical or until a predefined number of iterations

The idea of approximating the expected total remaining contribution (referred to as the "value function" or "cost-to-go function" in the dynamic programming literature) has a long history. More about historical developments and the various existing approaches can be found in monographs such as Powell (2007) or Sutton and Barto (1998). In particular, the idea of using extremely randomized trees as part of a dynamic programming approach has first been proposed by Ernst et al. (2005).

3 Case sudy: Application at a copper-gold deposit

This section presents an application of the proposed method using data from a copper-gold mine. The application is based on the case study of Goodfellow and Dimitrakopoulos (2013).

The mine complex under consideration consists of a single mine from which the extracted blocks can be sent to one of six initial destinations. The material extracted from the mine is comprised of three main groups (sulphides, transition and oxides), each of which is divided into two sub-groups, as illustrated o the left side of Figure 2. There are six destinations to which the material can be sent: a sulphide mill, a sulphide heap leach, a sulphide dump leach, a transition heap leach, an oxide heap leach and an oxide waste dump. The destinations are selective about the types of material they accept, as illustrated by the colour coding in Figure 2.

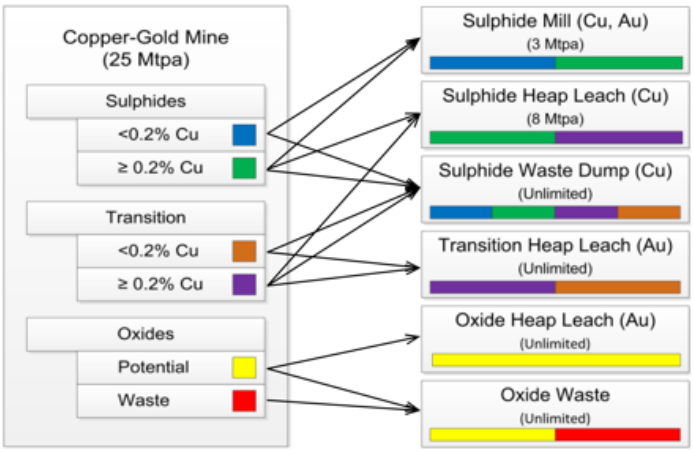

Figure 2: Graphical representation of the various material types and destinations. Note the color coding, representing the material types that can be accepted by each destination. Figure adapted from Goodfellow and Dimitrakopoulos (2013).

The sulphide mill is limited to processing three million tonnes per year, and produces both copper and gold outputs. The sulphide heap leach has an upper limit of eight million tonnes per year. The remaining destinations do not have any capacity constraints. All destinations, with the exception of the waste dump (which does not treat any material) have variable grade-recovery curves, as illustrated in Figure 3.

For this case study, only the first six mining periods (years) are considered. In addition, the study focuses on initial destination policies, therefore a pre-determined mining schedule is used. The mined blocks are clustered according to their properties, using the k-means++ (Arthur and Vassilvitskii, 2007) variant of the well-known k-means algorithm (Lloyd, 1982), This property-based clustering is similar to the methodology proposed by Goodfellow and Dimitrakopoulos (2012). Ten clusters are used for each material type, resulting

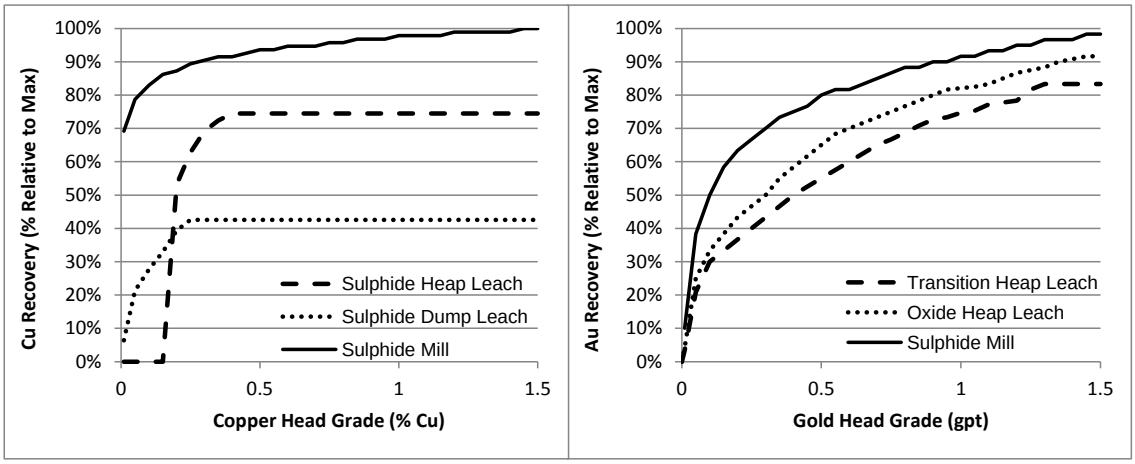

Figure 3: Grade-recovery curves for the various destinations. Recoveries expressed as percentage of maximum recovery for confidentiality purposes. Figure reproduced from Goodfellow and Dimitrakopoulos (2013).

in a total of 60 clusters. Note that this clustering scheme groups blocks according to similar properties, rather than according to geographical location. The remaining parameters used are identical to those in Goodfellow and Dimitrakopoulos (2013).

In order to model geological uncertainty, 50 orebody simulations are used. 40 of these simulations are used for generating training data for the regression tree ensemble, and the remaining 10 are used to test the robustness of the proposed approach. The initial policy used as part of the iterative policy improvement process is a greedy policy (a policy that maximizes only the local block contribution, and ignores the expected total remaining contribution). Due to computational limitations, the improvement process was set to terminate after a maximum of three iterations.

In order to write the objective function for this example, the following quantities are defined for each period t:

- $v^k_{MT}(t)$, $v^k_{Cu}(t)$ and $v^k_{Au}(t)$ are random variables denoting, for cluster $clus_k$, its total tonnage, copper tonnage, and gold tonnage, respectively; the 50 orebody simulations provide realizations of these random variables
- $in^{dest}_{MT}(t) = \sum_{k \ s.t. \ \pi(clus_k)=dest} v^k_{MT}(t)$ is the total tonnage of the material sent to destination $dest$; $in^{dest}_{Cu}(t)$ and $in^{dest}_{Au}(t)$ are the total tonnages for the copper and gold, respectively, sent to destination $dest$ in period t, and are computed similarly to $in^{dest}_{MT}(t)$
- $out^{dest}_{Cu}(t) = in^{dest}_{Cu}(t) \times Recovery^{dest}\left(in^{dest}_{Cu}(t),\ in^{dest}_{MT}(t)\right)$ is the total copper produced at $dest$ (for destinations that produce only gold, $out^{dest}_{Cu}(t) = 0$)
- $out^{dest}_{Au}(t) = in^{dest}_{Au}(t) \times Recovery^{dest}\left(in^{dest}_{Au}(t),\ in^{dest}_{MT}(t)\right)$ is the total gold produced at $dest$ (for destinations that produce only copper, $out^{dest}_{Au}(t) = 0$)
- p_{Cu} and p_{Au} are the selling prices for copper and gold, respectively
- c^{dest}_{proc} and c^{dest}_{sell} are the processing and selling costs, respectively, for $dest$
- pf^{dest} is the penalty factor for deviating from the capacity of destination $dest$
- LB^{dest} and UB^{dest} are the total tonnage lower and upper bound, respectively, for destination $dest$

The terms making up the objective function can now be written for this case study. As stated previously, the objective function has the general form:

$$Obj = \sum_{t=1}^{T} \frac{1}{(1+d)^t} \sum_{dest} E\left[Revenue^{dest}(t) - Cost^{dest}(t) - Penalty^{dest}(t)\right],$$

with the composing terms having the following values:

$$\begin{aligned} Revenue^{dest}(t) &= p_{Cu} \times out_{Cu}^{dest}(t) + \ p_{Au} \times out_{Au}^{dest}(t) \\ Cost^{dest}(t) &= \left(c_{proc}^{dest} + c_{sell}^{dest}\right) \times in_{MT}^{dest}(t) \\ Penalty^{dest}(t) &= pf^{dest} \times \left(\max\left(0,\ LB^{dest} - in_{MT}^{dest}(t)\right) + \max\left(0,\ in_{MT}^{dest}(t) - UB^{dest}\right)\right) \end{aligned}$$

3.1 Numerical results

Applying the policy improvement algorithm on this case study results in an improvement of **30.58%** in the value of the objective function after three iterations of the algorithm. This shows that the method works as intending, computing a tree-based per-state policy that improves upon the initial policy.

The reasons for the difference in the objective function can be deduced by examining the total tonnage of the materials sent to the different destinations, as well as the total value of the destinations' outputs. Since the first two destinations, the sulphide mill and the sulphide heap leach, generate most of the value, the total tonnage and output value graphs are only shown for these destinations.

The graph in Figure 4 shows the total tonnage sent to the mill. We can see that the tree-based policy does a much better job at matching the capacity of the mill than the greedy policy. This accounts for some of the difference in the objective function, since the solutions get penalized if they send less than 2.9 tons per year to the mill.

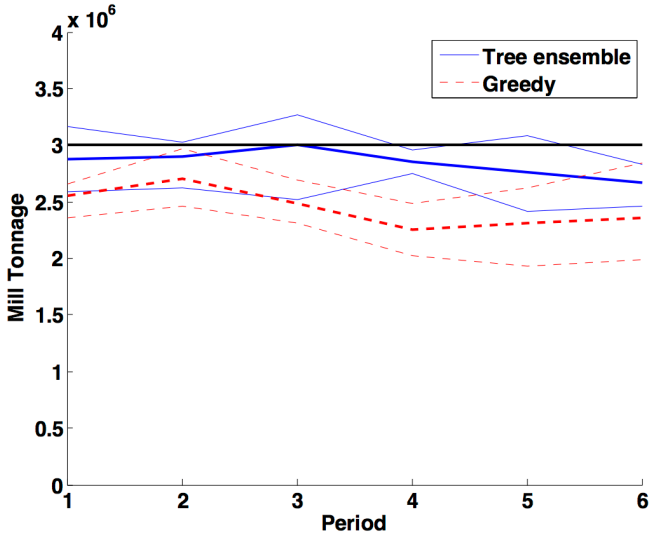

Figure 4: Total tonnage sent to the sulphide mill by the two policies. The horizontal black line corresponds to the mill's capacity (3 Mtpa). The thinner lines correspond to the 10th and 90th percentile (P10 and P90), while the thicker lines correspond to the averages.

The other main source of difference between the two policies appears to be given by the value of the copper produced by the mill. As can be seen in Figure 6, the total copper value is much higher for the tree-based policy than for the greedy policy. This appears to be because the greedy policy appears to choose to send some of the material with high copper grade to a less profitable destination (the heap leach) rather than to the mill. Indeed, it can be seen in Figure 8 that the total copper output at the heap leach is slightly

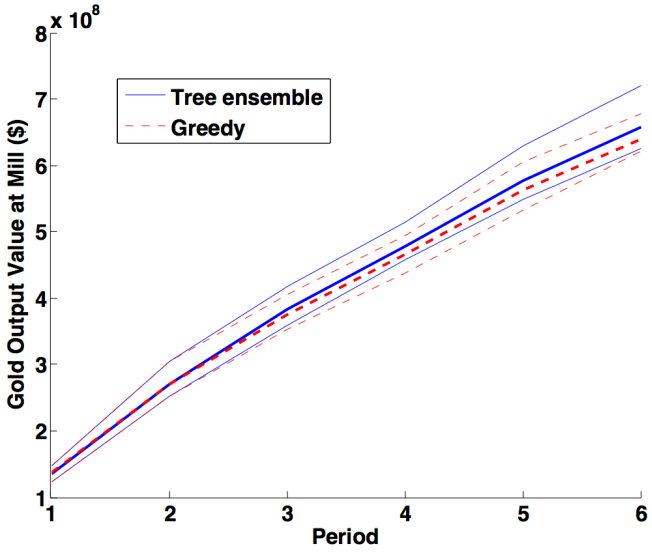

Figure 5: Total value of gold produced by the sulphide mill under the two policies. The thinner lines correspond to the 10th and 90th percentile (P10 and P90), while the thicker lines correspond to the averages.

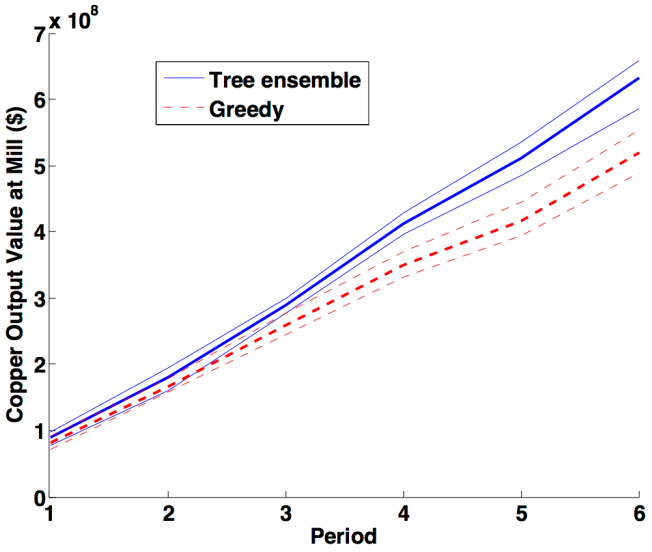

Figure 6: Total value of copper produced by the sulphide mill under the two policies. The thinner lines correspond to the 10th and 90th percentile (P10 and P90), while the thicker lines correspond to the averages.

higher for the greedy policy than for the tree-based policy. This may be explained by the fact that the greedy policy is trying to make sure that the lower bound on the heap leach capacity (7.8 tons) is met, so it sends some of the first blocks to be allocated to the heap leach rather than to the mill, despite their relatively high grade.

The other main source of difference between the two policies appears to be given by the value of the copper produced by the mill. As can be seen in Figure 6, the total copper value is much higher for the tree-based policy than for the greedy policy. This appears to be because the greedy policy appears to choose to send some of the material with high copper grade to a less profitable destination (the heap leach) rather than to the mill. Indeed, it can be seen in Figure 8 that the total copper output at the heap leach is slightly higher for the greedy policy than for the tree-based policy. This may be explained by the fact that the greedy

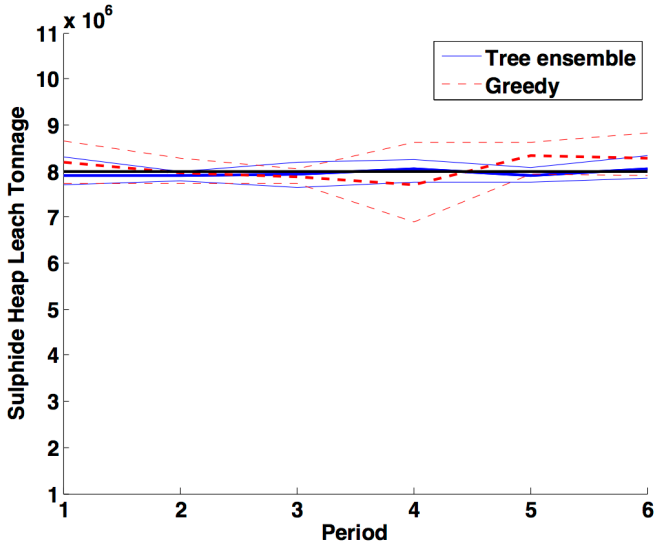

Figure 7: Total tonnage sent to the sulphide heap leach by the two policies. The horizontal black line corresponds to the capacity of the sulphide heap leach (8 Mtpa). The thinner lines correspond to the 10th and 90th percentile (P10 and P90), while the thicker lines correspond to the averages.

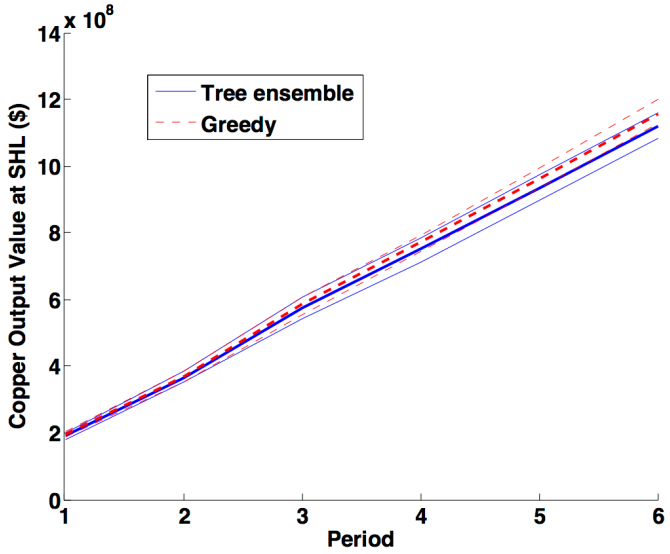

Figure 8: Total value of copper produced by the sulphide heap leach under the two policies. The thinner lines correspond to the 10th and 90th percentile (P10 and P90), while the thicker lines correspond to the averages.

policy is trying to make sure that the lower bound on the heap leach capacity (7.8 tons) is met, so it sends some of the first blocks to be allocated to the heap leach rather than to the mill, despite their relatively high grade.

Figures 5 and 7 show that the two policies are fairly similar in terms of total gold value produced by the mill and total tonnage sent to the heap leach.

4 Conclusions and future work

This paper introduces per-state policies as a new way to predict how a mining complex might adapt to a gradual reduction in uncertainty. It illustrates the concept on the problem of allocating extracted mineral to a set of initial destinations, and describes how approximate dynamic programming can be used in order to compute per-state policies.

The case study demonstrates that the approximate dynamic programming methodology can successfully construct a per-state policy with significantly superior performance than that of a policy based on a greedy heuristic. Analyzing how close the policy produced by our method is to an optimal policy remains a topic for future work.

Future work also needs to investigate how our method can be used for optimizing other parts of the mining complex, and show examples of its integration into a global optimization algorithm. Of particular interest is designing a realistic model of how the uncertainty is gradually reduced. In addition, computational improvements are needed in order to apply our method to more complex processing streams and to integrate with production scheduling.

References

Anderson, B. and Moore, J.B. (1989). Optimal Control. Prentice-Hall.

Arthur, D. and Vassilvitskii, S. (2007). k-means++ : The Advantages of Careful Seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, volume 8, 1027–1035.

Bellman, R. (1957). A Markovian decision process. Journal of Mathematics and Mechanics, 6, 679–684.

Bertsekas, D.P. and Tsitsiklis, J.N. (1996). Neuro-Dynamic Programming. Athena Scientific.

Birge, J. and Louveaux, F. (1997). Introduction to Stochastic Programming. Springer.

Boland, N., Dumitrescu, I., and Froyland, G. (2008). A Multistage Stochastic Programming Approach to Open Pit Mine Production Scheduling with Uncertain Geology. Draft Paper, available online at http://www.optimization-online.org/DB_FILE/2008/10/2123.pdf.

Breiman, L. (2001). Random forests. Machine Learning, 5–32.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1993). Classification and Regression Trees. Chapman and Hall.

Cormen, T H., Stein, C., Rivest, R.L., and Leierson, C.E. (2001). Introduction to Algorithms. MIT Press, 2nd edition.

Defourny, B., Ernst, D., and Wehenkel, L. (2012). Multistage Stochastic Programming: A Scenario Tree Based Approach to Planning under Uncertainty. In Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions, 97–143.

Del Castillo, F. and Dimitrakopoulos, R. (2013). Joint Effect of Commodity Price and Geological Uncertainty over the Life of Mine and Ultimate Pit Limit. Technical report, COSMO.

Dimitrakopoulos, R. (2011). Strategic Mine Planning Under Uncertainty. Journal of Mining Science, 47(2), 138–150.

Dimitrakopoulos, R., and Abdel Sabour, S.A. (2007). Evaluating mine plans under uncertainty: Can the real options make a difference? Resources Policy, 32(3), 116–125.

Dowd, P. (1976). Application of dynamic and stochastic programming to optimize cutoff grades and production rates. Transactions of the Institutions of Mining and Metallurgy, 1.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. Journal of Machine Learning Research, 6, 503–556.

Ernst, D., Glavic, M., Capitanescu, F., and Wehenkel, L. (2009). Control: A Comparison on a Power System Problem. IEEE Transactions on Systems, Man, and Cybernetics, 39(2), 517–529.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. Machine Learning, 63(1), 3–42.

Goodfellow, R. and Dimitrakopoulos, R. (2012). Mining Supply Chain Optimization under Geological Uncertainty. Technical report, COSMO.

Goodfellow, R. and Dimitrakopoulos, R. (2013). Global Asset Optimization of Open Pit Mining Complexes under Uncertainty. Technical report, COSMO.

Johnson, P., Evatt, G., Duck, P., and Howell, S. (2011). The determination of a dynamic cut-off grade for the mining industry. Lecture Notes in Electrical Engineering, 90, 391–403.

Kizilkale, A.C. and Dimitrakopoulos, R. (2013). Optimizing Mining Rates under Financial Uncertainty in Global Mining Complexes. Technical report, COSMO.

Lloyd, S. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2), 129–137.

Montiel, L. and Dimitrakopoulos, R. (2013). Optimizing Mining Complexes with Multiple Processing and Transportation Alternatives: An Uncertainty-Based Approach. Technical report, COSMO.

Powell, W. (2012). A unified framework for stochastic and dynamic programming. INFORMS Computing Society Newsletter.

Powell, W., Simao, H., and Bouzaiene-Ayari, B. (2012a). Approximate dynamic programming in transportation and logistics: A unified framework. European Journal of Transportation and Logistics, 1(3), 237–284.

Powell, W., George, A., Simao, H., Scott, W., Lamont, A., and Stewart, J. (2012b). Smart: A stochastic multiscale model for the analysis of energy resources, technology, and policy. INFORMS Journal on Computing, 24(4), 66–682.

Powell, W.B. (2007). Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley-Interscience.

Puterman, M.L. (1994). Markov Decision Processes: Discrete and Stochastic Dynamic Programming. Wiley, New York, NY.

Sutton, R.S. and Barto, A.G. (1998). Reinforcement Learning: An Introduction. MIT Press.

Wang, Q., Gu, X., and Chu, D. (2008). A dynamic optimization method for determining cutoff grades in underground mines. Gospodarka Surowcami Mineralnymi, 24(4/2), 133–142.

Whittle, G. (2007). Global Asset Optimisation. In Orebody modelling and strategic mine planning: Uncertainty and risk management models. AusIMM Spectrum Series 14, 331–336.

Whittle, G. (2010). Enterprise Optimisation. In Mine Planning and Equipment Selection Conference, 105–117.