ORIGINAL ARTICLE

The policy graph decomposition of multistage stochastic programming problems

Oscar Dowson

Correspondence
Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, 60208, IL
Email: oscar.dowson@northwestern.edu

Funding information

We propose the *policy graph* as a structured way of formulating a general class of multistage stochastic programming problems in a way that leads to a natural decomposition. We also propose an extension to the stochastic dual dynamic programming algorithm to solve a subset of problems formulated as a policy graph. This subset includes discrete-time, convex, infinite-horizon, multistage stochastic programming problems with continuous state and control variables. To demonstrate the utility of our algorithm, we solve an existing multistage stochastic programming problem from the literature based on pastoral dairy farming. We show that the finite-horizon model in the literature suffers from end-of-horizon effects, which we are able to overcome with an infinite-horizon model.

KEYWORDS
policy graph, multistage, stochastic programming, infinite horizon

1 | INTRODUCTION

This paper discusses the formulation of, and computation for, a class of multistage stochastic programming problems. Our first contribution is a structured framework for the practitioner to formulate multistage stochastic programming problems. We call this the *policy graph* framework. The nodes in a policy graph provide a natural means for decomposing the multistage stochastic program into a collection of subproblems, with arcs linking these subproblems representing the flow of information through time. The policy graph framework provides notational constructs that allow the practitioner to cleanly formulate and communicate a much broader class of multistage stochastic programming problems, relative to existing techniques.

Our second main contribution is an extension of the stochastic dual dynamic programming (SDDP) algorithm

(Pereira and Pinto, 1991) to solve a subset of models that can be formulated as a policy graph. This subset includes discrete-time, convex, infinite-horizon stochastic linear programming problems. While the corresponding multistage stochastic programs can have infinitely many scenarios, we assume that the policy graph has a modest number of nodes, which facilitates computation.

Unlike the standardization that has taken place in deterministic optimization (where terms like decision variable, constraint, and objective function are widely accepted), the stochastic optimization community has fragmented into different groups, each of which speaks a different arcane language (Powell, 2014). Therefore, in order to introduce the policy graph and explain our solution technique, it is necessary to clearly define the terminology and notation that we shall be using in this paper. Readers should be aware that compared with other approaches in the literature, there are some subtle (and some large) differences in our approach. We use the language of stochastic optimal control (Bertsekas, 2005), with terms like *stage*, *state*, and *control*. We do not view the world as a scenario tree. Nor do we explicitly express the uncertainty in terms of a filtration on some probability space. Instead, we follow an approach that is heavily inspired by the approximate dynamic programming framework of Powell (2011) and the Markov decision processes of Puterman (1994): we decompose the problem into small atomic pieces, clearly define each piece, and then define how the pieces fit together.

The paper is laid out as follows. In Section 2, we begin with some necessary terminology. Then, in Section 3, we introduce the *policy graph* as a way of formulating multistage stochastic programming problems. In Section 4, we present an algorithm to solve a subset of problems formulated as a policy graph. Finally, in Section 5 we demonstrate the utility of considering the infinite-horizon formulation of a problem arising from pastoral agriculture.

2 | TERMINOLOGY

2.1 | Basic definitions

First, let us define the *stage* in multistage.

Definition A *stage* is a discrete moment in time in which the agent chooses a decision and any uncertainty is revealed.

Therefore, *multistage* refers to a problem that can be decomposed into a sequence of stages. This requires the assumption that time can be discretised. Second, the *stochastic* component of multistage stochastic programming refers to problems with uncertainty. In this paper, we differentiate between two types of uncertainty; the first of which we term a *noise*. (We shall describe the second type in the next section, but it relates to how the problem transitions between stages.)

Definition A *noise* is a stagewise-independent random variable in stage t.

In stage t, we denote a single observation of the noise with the lowercase ω_t, and the sample space from which it is drawn by the uppercase Ω_t. Ω_t can be continuous or discrete, although in this paper we only consider the discrete case. Furthermore, we use the term *stagewise-independent* to refer to the fact that the distribution of the noise in stage t is independent of the noise in other time periods. $1, 2, \dots, t-1, t+1, t+2, \dots$.

Next we define a *state*, modifying slightly the definition from Powell (2016).

Definition A *state* is a function of history that captures all the information we need to model a system from some point in time onward.

Expressed a different way, a state is the smallest piece of information that is necessary to pass between stage t and $t + 1$ so that the optimal decision-making in stage $t + 1$ onward can be made independently from the decisions that were made in stages 1 to t. Each dimension of the state is represented by a *state variable*. State variables can be continuous or discrete.

We denote the state variable at the start of stage t by the lowercase x_t. We refer to x_t as the *incoming state variable*. Then, during the stage: the agent chooses a control (action); a realization of the noise is observed; and the state transitions to x'_t at the end of the stage according to the transition function. We refer to x'_t as the *outgoing state variable*. We now define a *control* and the *transition function*.

Definition A *control* variable is an action or decision taken (explicitly or implicitly) by the agent during a stage.

Control variables in stage t can be continuous or discrete. In this paper, we denote control variables with the lowercase u_t. Such controls must be feasible for the agent to implement, and therefore they belong to a set that depends on the incoming state variable and observation of the random noise; this is denoted $u_t \in U_t(x_t, \omega_t)$.

Definition The *transition function* is a mapping of the incoming state x_t to the outgoing state x'_t, given the control u_t and the noise ω_t.

We denote the transition function as $x'_t = T_t(x_t, u_t, \omega_t)$. This function can be of any form. As a result of the state transitioning, a cost is incurred.

Definition The *stage-objective* is the cost (if minimizing, otherwise value) accrued in stage t as a consequence of taking the control u_t, given the incoming state x_t and realization of the noise ω_t.

This is denoted $C_t(x_t, u_t, \omega_t)$. All that remains is to define how the agent chooses a control. We use the terminology of Puterman (1994) and call this a *decision-rule*.

Definition A *decision-rule* π_t, for stage t, is a mapping of the incoming state variable x_t and observation of the noise ω_t to a control u_t.

This is denoted $u_t = \pi_t(x_t, \omega_t)$. In cases where the decision-rule does not depend upon the noise, we denote the decision-rule as $\pi_t(x_t)$. We refer to a set of decision-rules, one for each stage t, as a *policy*.

Definition A *policy* is a set of decision-rules $\pi = \{\pi_t : t = 1, 2, \ldots, T\}$, containing one element for each stage t.

2.2 | Nodes

Now that we have defined some basic terminology, we can construct the atomic building block of multistage stochastic programming, which we refer to as a *node*.

Definition A *node* is a collection of the following components: incoming and outgoing state variables, a noise, some control variables, a transition function, a stage-objective, and a decision-rule.

We discriminate between two types of node: *Hazard-Decision* and *Decision-Hazard*.

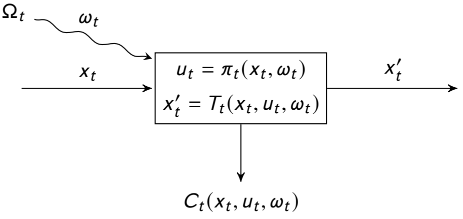
Ω_t
ω_t
x_t
$u_t = \pi_t(x_t, \omega_t)$
$x'_t = T_t(x_t, u_t, \omega_t)$
x'_t
$C_t(x_t, u_t, \omega_t)$

FIGURE 1 Schematic of a Hazard-Decision node.

Definition In a *Hazard-Decision* node, the agent chooses a control u_t *after* observing a realization of the noise $\omega_t \in \Omega_t$ according to the decision-rule $\pi_t(x_t, \omega_t)$. The state transitions from x_t to x'_t according to the transition function $T_t(x_t, u_t, \omega_t)$. The decision-rule respects the set of admissible controls so that $u_t \in U_t(x_t, \omega_t)$. In addition, a cost $C_t(x_t, u_t, \omega_t)$ is incurred. A schematic of this is shown in Figure 1.

Definition In a *Decision-Hazard* node, the agent chooses a control u_t *before* observing a realization of the noise $\omega_t \in \Omega_t$ according to the decision-rule $\pi_t(x_t)$. The state transitions from x_t to x'_t according to the transition function $T_t(x_t, u_t, \omega_t)$. The decision-rule respects the set of admissible controls so that $u_t \in U_t(x_t)$. In addition, a cost $C_t(x_t, u_t, \omega_t)$ is incurred. A schematic of this is shown in Figure 2.

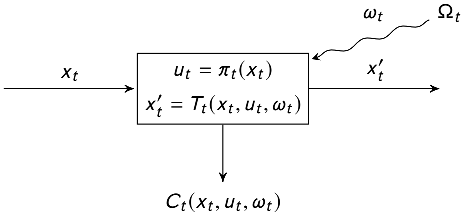

FIGURE 2 Schematic of a Decision-Hazard node.

Readers should note that the set of admissible controls $U_t(x_t)$ must be such that u_t is feasible for every possible realization of ω_t. Therefore, $U_t(x_t)$ can be thought of as the intersection of the feasible set for each realization of ω, i.e., $U_t(x_t) = \cap_{\omega_t \in \Omega_t} U_t(x_t, \omega_t)$.

When no uncertainty is realized in the node (i.e., $|\Omega_t| = 1$), the Decision-Hazard node becomes identical to the Hazard-Decision node. We denote this deterministic node by dropping the wavy line coming into the box depicting the node and dropping the ω_t function arguments. Furthermore, readers should note that the two types of nodes are not immutable. Instead, they are a modelling choice. For example, it is possible to transform a Decision-Hazard node into a deterministic node followed by a Hazard-Decision node with an expanded state-space (see, e.g., Street et al. (2018)). An example of this is shown in Figure 3.

In the left-hand node of Figure 3, we seek a decision rule $u_t = \pi_t(x_t)$, which is identical to the decision rule of the Decision-Hazard node. There is no need for a transition function; instead, we just pass through x_t unchanged. Since the control u_t is needed in the next node, it also gets passed along as a temporary state variable. In addition, no costs are incurred. In the second node, the uncertainty is realized. After the uncertainty is realized, the state transitions from x_t to x'_t according to the same transition function as the Decision-Hazard node. There is no need to compute a

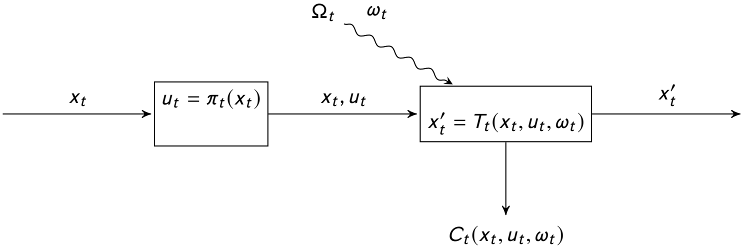

FIGURE 3 A Decision-Hazard node expanded into a deterministic node and a Hazard-Decision node.

decision rule because the control u_t has already been chosen. Finally, a cost of $C_t(x_t, u_t, \omega_t)$ is incurred, where C_t is the stage-objective of the original Decision-Hazard node.

Despite the fact that this example demonstrates that the concept of a Decision-Hazard node is superfluous, we retain the distinction between Decision-Hazard and Hazard-Decision because it is a useful modelling device. However, as shown in Street et al. (2018), the Decision-Hazard formulation is typically much more difficult to solve than the Hazard-Decision formulation, and so the first step in any solution process is to convert the problem into a Hazard-Decision formulation.

2.3 | Example

At this point, it is useful to consider a small example in order to contextualize the terminology we have been discussing. As our example, we choose a small portfolio optimization problem with two investment classes to choose from: stocks and bonds.

In our example, there are three stages, which we denote $t = 1, 2, 3$. Since the value of stocks and bonds we own in stage t depends on the value we owned in stage $t - 1$, these are *state variables*. We denote the value of stocks owned at the start of stage t by x_t^s, and the value of bonds by x_t^b. For control variables, in each stage t we can buy or sell stocks, denoted u_t^s, and bonds, denoted u_t^b. In stage t, the *noise* is the market return, which we denote ω_t^s for stocks and ω_t^b for bonds. The market returns are distributed as follows:

$$\Omega_t = \begin{cases} (\omega_t^s, \omega_t^b) = (1.1, 1.05) & w.p.\ 0.75 \\ (\omega_t^s, \omega_t^b) = (0.9, 0.95) & w.p.\ 0.25 \end{cases}$$

In the Hazard-Decision setting, the agent re-balances their portfolio *after* observing the market return ω_t. In other words, the agent re-balances their portfolio immediately prior to market close at the end of each day. To do so, they need a decision rule $u_t = \pi_t(x_t, \omega_t)$. Thus, for the feasibility set $U_t(x_t, \omega_t)$, we have constraints that the value of stocks and bonds owned must be non-negative, and we need an "conservation of value" constraint, so:

$$U_t(x_t, \omega_t) = \left\{ u_t \in \mathbb{R}^2 : \begin{array}{r} u_t^s + u_t^b = 0 \\ \omega_t^s x_t^s + u_t^s \geq 0 \\ \omega_t^b x_t^b + u_t^b \geq 0 \end{array} \right\}.$$

The transition function T is trivial: the value of stocks (resp. bonds) at the start of stage $t + 1$ is equal to the value of stocks (resp. bonds) at the start of stage t multiplied by the market return plus the purchase (or sale) u_t^s (resp. u_t^b). Thus, we have:

$$\begin{bmatrix} x_t^{s\prime} \\ x_t^{b\prime} \end{bmatrix} = T_t(x_t, u_t, \omega_t) = \begin{bmatrix} \omega_t^s x_t^s + u_t^s \\ \omega_t^b x_t^b + u_t^b \end{bmatrix}.$$

In the first two stages, the stage-objective is 0 (i.e., we assume that there are no transaction costs). In the final stage $t = 3$, the stage-objective is:

$$C_3(x_3, u_3, \omega_3) = \omega_3^s \cdot x_3^s + \omega_3^b \cdot x_3^b.$$

In the Decision-Hazard setting, the agent re-balances their portfolio *before* observing the market return. In other words, the agent re-balances their portfolio immediately after the market open at the start of each day. To do so, they need a decision rule $u_t = \pi_t(x_t)$. The corresponding feasibility set U_t is as follows:

$$U_t(x_t) = \left\{ u_t \in \mathbb{R}^2 : \begin{array}{r} u_t^s + u_t^b = 0 \\ x_t^s + u_t^s \geq 0 \\ x_t^b + u_t^b \geq 0 \end{array} \right\}.$$

We also have to account for the change in the order of the decision making, so the transition function is:

$$\begin{bmatrix} x_t^{s\prime} \\ x_t^{b\prime} \end{bmatrix} = T_t(x_t, u_t, \omega_t) = \begin{bmatrix} \omega_t^s(x_t^s + u_t^s) \\ \omega_t^b(x_t^b + u_t^b) \end{bmatrix}.$$

Finally, in the final stage $t = 3$, the stage-objective is:

$$C_3(x_3, u_3, \omega_3) = \omega_3^s \cdot (x_3^s + u_3^s) + \omega_3^b \cdot (x_3^b + u_3^b).$$

3 | POLICY GRAPHS

The definitions in the previous section should be familiar to most readers versed in stochastic programming. However, notably excluded is a description of *how* the nodes are linked together. Typically, nodes are linked together as a linear sequence so that $x_t' = x_{t+1}$. If so, we use the terms *node* and *stage* interchangeably, and we use x_t and x_{t+1} instead of x_t and x_t'. Implicit within the sequential linking of nodes is the idea that u_t cannot depend upon events in stages $t + 1$ onward (i.e., nonanticipativity), since the decision-rule π_t depends only upon the incoming state variable x_t and the stagewise independent random variable ω_t.

In contrast to the linear case, it is possible to link nodes together in more complicated structures. In these structures, there can be many nodes corresponding to each stage (in time). Therefore, we cannot use the terms *node* and *stage* interchangeably. To clearly signal the difference, we will use the subscript i instead of t when referring to the components within each node. Moreover, for each of the previous definitions, the reader should replace all references to "stage t" with "node i." Additionally, whenever we say "stagewise-independent," we really mean "nodewise-independent." More concretely, we mean the noise in node i, ω_i, is independent of the noise at all other nodes. However, we shall retain the terminology of "stagewise" independence since it is a term commonly used in the literature.

In the linear case, the sequence of decision making flows from stage t to stage $t+1$. In more complicated graph structures, we generalize this sequence into a Markov process between nodes in the graph. This is the second type of uncertainty that alluded to in the previous section. (Recall the first type of uncertainty was the stagewise-independent *noise* within a node.) We denote the matrix of transition probabilities by Φ, with elements $\phi_{i,j}$ denoting the probability of transitioning from node i to node j. In the linear case described above, the Markov process is trivial: the probability of transitioning from stage t to stage $t+1$, $\phi_{t,t+1}$, is 1, and the probability of transitioning from stage t to any other stage t' is 0.

We now have almost all the terminology necessary to define a *policy graph*. Before we do, we need to define the initial conditions of the problem.

Definition The *root node* is the current point-of-view of the agent in the decision-making process and stores an initial value x_R for the state variable.

It is important to note that the root node is neither a Hazard-Decision nor a Decision-Hazard node; it is just a convenient object to represent the initial point in the sequential decision-making process.

Definition A *policy graph* $\mathcal{G} = (R, \mathcal{N}, \mathcal{E}, \Phi)$ is defined by a tuple containing the root node R, along with the set of nodes $\mathcal{N}$ and directed edges $\mathcal{E}$. Φ is an $|\mathcal{N}|+1$ by $|\mathcal{N}|$ matrix of the transition probabilities between nodes, with entries $\phi_{i,j}$, such that if there exists an edge (i,j) in $\mathcal{E}$ for two nodes $i \in \mathcal{N} \cup \{R\}$ and $j \in \mathcal{N}$, then $x_j = x_i'$ with probability $\phi_{i,j} > 0$. If no edge exists, then $\phi_{i,j} = 0$.

It is also useful to define the *children* of a node and the notion of two nodes being *connected*.

Definition The *children* of node i are the nodes in the set $i^+ = \{j : \phi_{i,j} > 0\}$.

Definition Node i is *connected* to node j if $j \in i^+$.

Definition Node i is a *leaf* node if $i^+ = \varnothing$.

Returning to our example from Section 2.3, the policy graph can be defined as follows:

$$\mathcal{G} = (R, \mathcal{N}, \mathcal{E}, \Phi) = (t_0, \{t_1, t_2, t_3\}, \{(t_0, t_1), (t_1, t_2), (t_2, t_3)\}, \{\phi_{t_0,t_1} = 1, \phi_{t_1,t_2} = 1, \phi_{t_2,t_3} = 1\}),$$

where the nodes t_1, t_2, and t_3 are defined as outlined in Section 2.3. Writing out the explicit formulation of the policy graph is tedious, especially if there are a large number of nodes. Instead, it is much easier to convey the structure of a policy graph using a graphical representation. Figure 4 shows a hazard-decision formulation of the portfolio problem, and Figure 5 shows a decision-hazard formulation. Each figure has a single circle that represents the root node. The square boxes represent Hazard-Decision or Decision-Hazard nodes. For simplicity, we drop the annotations used in Figures 1 and 2, since the node type can be inferred based on the presence (or absence) of a wavy arc representing the noise.

We call policy graphs with a structure like Figures 4 and 5 *linear policy graphs*.

Definition A *linear policy graph* is composed of a finite set of nodes, where each node is connected to, at most, one other node.

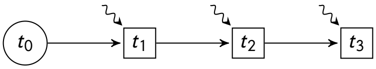

FIGURE 4 Portfolio allocation problem: Hazard-Decision formulation.

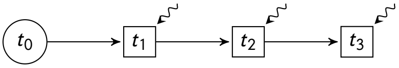

FIGURE 5 Portfolio allocation problem: Decision-Hazard formulation.

When the policy graph is linear, we can use the terms *node* and *stage* interchangeably, and we denote the outgoing state variable in stage t as x_{t+1} instead of x_t'.

We note that this paper is not the first to advocate for a graphical representation of the decision-making process. The System Dynamics community have developed a rich graphical framework for conveying dynamical systems (Sterman, 2000). However, the System Dynamics approach focuses on the (causal) relationships between the states (stocks) and controls (flows) and cannot express how uncertainty is revealed through the decision-making process. There are also well-known tools in the Operations Research community that use similar symbols such as decision trees, scenario trees, and flowcharts (see Powell (2011) for examples). Our approach is different in that it operates at a level of abstraction above the System Dynamics framework (for example, each node can be diagrammed using System Dynamics), and that (as we shall show) it supersedes the scenario tree approach.

To show that policy graphs can also describe scenario trees, consider the following policy graph (also shown in Figure 6):

$$\mathcal{G} = (R, \mathcal{N}, \mathcal{E}, \Phi) = \left(R, \{B, C\}, \{(R, B), (B, C)\}, \{\phi_{R,B} = 1, \phi_{B,C} = 1\}\right).$$

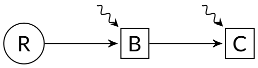

FIGURE 6 Two-stage Hazard-Decision problem.

If each Hazard-Decision node in Figure 6 has two possible realizations ω_1 and ω_2, with probabilities p_1 and p_2 respectively, then it can be expanded into the scenario tree with six nodes. This can be described by the policy graph:

$$\begin{aligned}\mathcal{G} = \quad & (R, \mathcal{N}, \mathcal{E}, \Phi) = (\\ & R, \\ & \{B_1, B_2, C_{11}, C_{12}, C_{21}, C_{22}\}, \\ & \{(R, B_1), (R, B_2), (B_1, C_{11}), (B_1, C_{12}), (B_2, C_{21}), (B_2, C_{22})\}, \\ & \{\phi_{R,B_1} = p_1, \phi_{R,B_2} = p_2, \phi_{B_1,C_{11}=p_1,\phi_{B_1,C_{12}}=p_2,\phi_{B_2,C_{21}}=p_1,\phi_{B_2,C_{22}}=p_2}\} \\ ). & \end{aligned}$$

The graphical representation is shown in Figure 7. Note that every node is now deterministic (there are no wavy lines in the graph). This is because we have moved the uncertainty from the stagewise-independent noise terms ω to the node-transition matrix Φ.

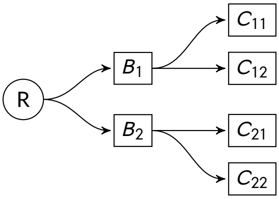

FIGURE 7 A policy graph that is equivalent to a scenario tree.

Formulating a multistage stochastic programming model as a scenario tree offers significant generality (e.g., we are no longer constrained to stagewise-independent noise), but may come at the cost of decreased computational tractability (there are more nodes, and therefore more decision-rules to compute).

The benefit of the graphical representation of a policy graph is most apparent when different types of nodes are mixed together. For example, a common problem in the literature is the two-stage problem with recourse (Figure 8). In this problem, the first stage is deterministic, and the second stage is Hazard-Decision. Alternatively, it is possible to formulate the problem as shown in Figure 9: the first stage is Decision-Hazard, and the second stage is deterministic.

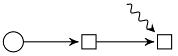

FIGURE 8 Two-stage problem with recourse: hazard-decision.

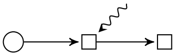

FIGURE 9 Two-stage problem with recourse: decision-hazard.

3.1 | Stagewise-dependent processes

So far, we have limited the uncertainty to two types: (i) a stagewise-independent *noise* within a node; and (ii) Markovian transitions between nodes. Importantly, the transition between the nodes determines the sample space from which the stagewise-independent noise is sampled. We now show how these two types of uncertainty can be combined to produce stagewise-dependent noise (just like stagewise-independent noise, we really mean *nodewise*-dependent noise). (This is not the only way to model stagewise-dependent noise terms. Another way is through a state-space expansion, see, e.g., Guigues (2014).) As a motivating example, consider a multistage stochastic programming model set in New Zealand with rainfall as a random variable (e.g., Dowson et al. (2019)). The New Zealand summer climate is dominated by the El Niño-Southern Oscillation, an irregular, periodical climate pattern in the Pacific Ocean. It has two extremes: El Niño and La Niña. In El Niño years, the Eastern Pacific warms relative to average, and there is less rainfall than average in New Zealand. In La Niña years, the opposite is true (NIWA, 2018). In addition, assume that the sequence of El Niño and La Niña years can be modelled by a Markov process.

We could model this problem by a linear policy graph like Figure 8. However, we would need to add a binary state variable to code whether the system was in El Niño or La Niña, encode the Markovian process into the transition function T, and importantly, model the rainfall using only a stagewise-independent noise. Also note that for each stage, there would be one decision-rule.

Alternatively, we can model the problem as shown in Figure 10 without the need for the additional binary state variable. In addition, the distribution of the noise (rainfall) can differ between nodes, and there is no restriction on the relationship between the distribution of rainfall in El Niño years and the distribution of rainfall in La Niña years. Note that this formulation comprises of two decision-rules for each stage: one to use if the year is El Niño and one to use if the year is La Niña.

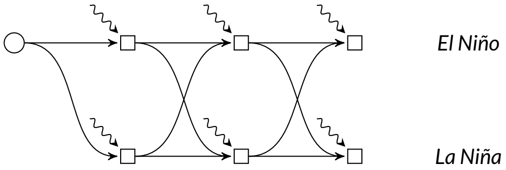

FIGURE 10 A Markovian policy graph.

Importantly, the realization of the random node transition (into either El Niño or La Niña) is exogenous to the remainder of the problem. Such a decomposition is hardly novel – indeed it is identical to the process of decomposing the problem into stages (and thereby solving for a decision-rule in each stage, instead of a single decision-rule that includes time as a state variable). We call policy graphs with this structure, where all the nodes in a stage t are connected to all the nodes in the next stage $t + 1$, *Markovian* policy graphs, and we refer to the different nodes within a stage as the *Markov states* of the stage.

Definition A *Markovian policy graph* is composed of a finite set of stages, where each stage consists of a set of nodes, and all the nodes within a stage are connected to nodes within, at most, one other stage.

The scenario tree in Figure 7 is also a Markovian policy graph. However, there are some structures which are neither linear nor Markovian; an example of which is shown in Figure 11. In that example, the policy graph is not Markovian as the node in the first stage has one child in the second stage and one in the third stage. Variations of these acyclic policy graphs are well studied in the work of Rebennack (2016).

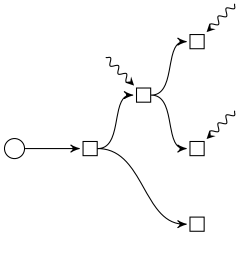

FIGURE 11 A non-Markovian policy graph.

Non-Markovian policy graphs reinforce the need to distinguish between a node and a stage. However, once this distinction is made, we can easily express problems with a random number of stages such as those considered by Guigues (2018). An example is shown in Figure 12. In this policy graph, it appears that there are a random number of stages because the agent may make two, three, or four decisions depending on which path they take through the graph.

This example is a good demonstration of how the policy graph approach is able to generalize existing special cases in the literature.

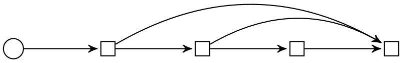

FIGURE 12 A policy graph with random transitions between stages.

3.2 | Cyclic policy graphs

The examples above were all acyclic. However, the policy graph approach naturally extends to infinite-horizon problems. For example, Figure 13 shows an infinite-horizon, Decision-Hazard problem with some probability of exiting the cycle into a final recourse node.

One approach to solving infinite-horizon problems is to incorporate a *discount factor* in order to produce a finite sum of the future costs (assuming, of course, a bounded stage-objective); this can be represented in a policy graph by allowing the probabilities of the arcs exiting a stage to sum to less than one. Implicitly, this results in the addition of a leaf node with zero cost (Figure 14). The discount factor ρ can be interpreted as a $1 - \rho$ probability that the system stops (Puterman, 1994; Bertsekas, 2005). Sometimes, it is more natural to express the discount factor in terms of an interest rate r, such that $\rho = 1/(1 + r)$.

It is important to reiterate that the transition between nodes is random and exogenous to the rest of the problem (i.e., the states, controls, and noise). The two diagrams do *not* represent stopping problems where the agent can make the decision to exit the cycle. Instead, stopping problems can be modelled with a binary state variable representing whether the agent is stopped or not.

We are almost ready to describe the optimization problem of finding the optimal decision-rule for each node. However, before we proceed, it is necessary to introduce the concept of *risk*.

3.3 | Risk

At every node in the policy graph of a multistage stochastic programming problem, the agent needs some way of aggregating the future cost of a control across the future uncertainty. They do so via a *risk measure*.

Definition A *risk measure* $\mathbb{F}$ is a function that maps a random variable to a real number.

To elaborate upon this definition, we draw heavily from Shapiro et al. (2009, Ch. 6.3). In this paper, we restrict our attention to random variables with a finite sample space $\Omega := \{z_1, z_2, \ldots, z_K\}$ equipped with a sigma algebra of all subsets of Ω and respective (strictly positive) probabilities $\{p_1, p_2, \ldots, p_K\}$. This greatly simplifies the analysis of risk and is a required assumption for our proposed solution technique (SDDP). We denote the random variable with the uppercase Z.

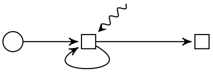

FIGURE 13 A cyclic policy graph.

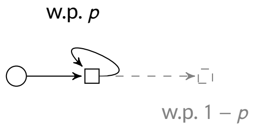

FIGURE 14 Cyclic policy graph with implicit leaf node.

In this paper, we shall use the following risk measures: Expectation ($\mathbb{E}[Z]$), Average Value-at-Risk ($\text{AV@R}_{1-\beta}[Z]$), and Worst-case ($\max[Z]$). The Expectation and Worst-case risk measures are self-explanatory. However, the Average Value-at-Risk is worth explaining for readers unfamiliar with it.

Definition According to Rockafellar and Uryasev (2002), the *Average Value-at-Risk*[1] at the β quantile ($\text{AV@R}_{1-\beta}$) is:

$$\text{AV@R}_{1-\beta}[Z] = \inf_{\zeta}\left\{\zeta + \frac{1}{\beta}\sum_{k=1}^{K} p_k(z_k - \zeta)_+\right\},$$

where $(x)_+ = \max\{0, x\}$.

As a simple approximation, the $\text{AV@R}_{1-\beta}$ can be thought of as the expectation of the worst β fraction of outcomes. However, if the distribution of the random variable is not continuous (e.g., the distribution is discrete), the interpretation is subtler since we may have to split a discrete probability atom (see Rockafellar and Uryasev (2002) for more details). Also note that when $\beta = 1$, $\text{AV@R}_{1-\beta}[Z] = \mathbb{E}[Z]$, and $\lim_{\beta\rightarrow 0}\text{AV@R}_{1-\beta}[Z] = \max[Z]$.

We use these risk measures (Expectation, AV@R, and Worst-case) because they are *coherent* according to the following axioms.

Definition A *coherent* risk measure is a risk measure $\mathbb{F}$ that satisfies the axioms of Artzner et al. (1999). For two discrete random variables Z_1 and Z_2, each with drawn from a sample space with K elements, the axioms are:

- **Monotonicity**: If $Z_1 \leq Z_2$, then $\mathbb{F}[Z_1] \leq \mathbb{F}[Z_2]$.
- **Sub-additivity**: For Z_1, Z_2, then $\mathbb{F}[Z_1 + Z_2] \leq \mathbb{F}[Z_1] + \mathbb{F}[Z_2]$.
- **Positive homogeneity**: If $\lambda \geq 0$ then $\mathbb{F}[\lambda Z] = \lambda\,\mathbb{F}[Z]$.
- **Translation equivariance**: If $a \in \mathbb{R}$ then $\mathbb{F}[Z + a] = \mathbb{F}[Z] + a$.

We can also define coherent risk measures in terms of *risk sets* (Artzner et al., 1999; Shapiro et al., 2009). That is, a coherent risk measure $\mathbb{F}$ has a dual representation that can be viewed as taking the expectation of the random variable with respect to the worst probability distribution within some set $\mathfrak{A}$ of possible distributions:

$$\mathbb{F}[Z] = \sup_{\xi\in\mathfrak{A}} \mathbb{E}_{\xi}[Z] = \sup_{\xi\in\mathfrak{A}} \sum_{k=1}^{K} \xi_k z_k, \tag{1}$$

where $\mathfrak{A}$ is a convex subset of:

$$\mathfrak{P} = \left\{\xi \in \mathbb{R}^K \ :\ \sum_{k=1}^{K}\xi_k = 1,\ \xi \geq 0\right\}.$$

[1] Rockafellar and Uryasev (2002) actually call this the *Conditional Value-at-Risk* (CV@R); however, we follow Shapiro et al. (2009) and refer to it as AV@R.

Following Philpott et al. (2013), we shall refer to the probability distribution ξ that attains the supremum of (1) as the *changed* probability distribution.

The three risk measures described above (Expectation, AV@R, and Worst-case) can be expressed in terms of the set $\mathfrak{A}$ as follows:

- **Expectation:** If $\mathfrak{A}$ is a singleton, containing only the original probability distribution, then the risk measure $\mathbb{F}$ is equivalent to the expectation operator.
- **AV@R:** If $\mathfrak{A} = \left\{\xi \in \mathfrak{P} \mid \xi_k \leq \frac{p_k}{\beta},\ k = 1, 2, \ldots, K\right\}$, then the risk measure $\mathbb{F}$ is equivalent to AV@R$_{1-\beta}$.
- **Worst-case:** If $\mathfrak{A} = \mathfrak{P}$, then $\mathbb{F}$ is the Worst-case risk measure.

3.4 | Standard form

Let us recap what we have discussed so far. We can decompose a multistage stochastic programming problem into a series of nodes. The nodes are linked together by state variables, and we can describe the linkages by a policy graph $\mathcal{G}$. Associated with each node i is a decision-rule $\pi_i(x_i, \omega_i)$ ($\pi_i(x_i)$ for Decision-Hazard), which maps the incoming state variable x_i and realization of a random noise ω_i, to a feasible control $u_i \in U_i(x_i, \omega_i)$ ($u_i \in U_i(x_i)$ for Decision-Hazard). As a result of taking the control u_i, the state transitions to the outgoing state x_i' according to the transition function $x_i' = T_i(x_i, u_i, \omega_i)$, and a cost of $C_i(x_i, u_i, \omega_i)$ is incurred. Then, the system transitions to a new node in the policy graph according to the Markov transition matrix Φ. All that remains is to define an optimization problem that can be used to find the optimal decision-rule π_i for each node i. To do this, we utilize Bellman's *principle of optimality* and form the *cost-to-go* for each node i, given the incoming state x_i and realization of the noise ω_i.

Definition *Principle of Optimality*: An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions. (Bellman, 1954)

Definition Given a policy π, the *cost-to-go* for node i is the stage-objective as a result of taking the control u_i, plus the risk-adjusted cost-to-go of the node's children:

$$V_i(x_i, \omega_i) = C_i(x_i, u_i, \omega_i) + \underset{j \in i^+;\, \omega_j \in \Omega_j}{\mathbb{F}_{\mathrm{i}}} \left[V_j(T_i(x_i, u_i, \omega_i), \omega_j)\right],$$

where $u_i = \pi_i(x_i, \omega_i)$ if the node is Hazard-Decision, and $u_i = \pi_i(x_i)$ if the node is Decision-Hazard.

Note that when we use the notation $\mathbb{F}_{j \in i^+;\, \omega_j \in \Omega_j}[\cdot]$, we mean that we apply the risk measure to the distribution of outcomes for all nodes $j \in i^+$ and realizations $\omega_j \in \Omega_j$. Since the noise realization and node transition are independent, from node i, the probability of realizing ω_j in the next node is $\phi_{i,j} \times p_{\omega_j}$ (i.e., the probability of transitioning from node i to node j, multiplied by the probability of realizing ω_j in node j). Also note that when maximizing, we refer to the *value-to-go* instead of the cost-to-go.

Putting together all that we have discussed so far, we can now define the class of multistage stochastic programming problems that we consider in the remainder of this paper.

Definition Given a policy graph $\mathcal{G}$, a *multistage stochastic programming problem* is an optimization problem of the form:

$$\min_{\pi} \left\{ \underset{i \in R^+;\, \omega_i \in \Omega_i}{\mathbb{F}} \left[V_i(x_R, \omega_i)\right] \right\}, \tag{2}$$

where x_R is the initial condition of the state variable at the root node.

Readers should note that in the interests of generality, we have purposefully avoided a definition of the spaces on which C_i, T_i, and U_i are defined. For now, we just assume that (2) is well-posed, and we will return to this question in the next section. Moreover, readers should note that our definition does not encompass all multistage stochastic programming problems. For example, it does not consider continuous time problems. In addition, for the algorithm we propose we will require further assumptions that we make precise in the next section.

Although each $\mathbb{F}$is a single-period risk measure of the type discussed in Section 3.3, the nested formulation of the cost-to-go function V creates a conditional risk mapping (Ruszczyński and Shapiro, 2006). However, our since our framework allows the single-stage risk measures to differ between nodes, one can model expected conditional risk measures (Homem-de-Mello and Pagnoncelli, 2016), and end-of-horizon risk measures by dynamically changing the single-period risk measures in each subproblem (Pflug and Pichler, 2016; Baucke et al., 2018). Moreover, because the risk measure is applied locally to each node in a nested fashion, the property of *time consistency* naturally arises. (A proper discussion on time-consistency is out-of-scope for the current paper, see, e.g., Shapiro et al. (2009); Homem-de-Mello and Pagnoncelli (2016); Valladão et al. (2019).) Finally, note that the non-anticipative constraints usually associated with multistage stochastic programming are satisfied by our recursive definition of a policy.

The goal of multistage stochastic programming is to find the policy π that solves (2). If the policy π is not given explicitly, the decision rule for node i can be formulated as an optimization problem. The formulation is different depending upon whether the node is Hazard-Decision or Decision-Hazard. We call the optimization problem associated with each node a *subproblem*.

Definition A *Hazard-Decision subproblem* is the optimization problem:

$$\begin{aligned}\mathbf{HD}_i(x_i,\omega_i):\quad V_i(x_i,\omega_i) = \min_{u_i,\bar{x}_i,x_i'} \quad & C_i(\bar{x}_i,u_i,\omega_i) + \underset{j\in i^+;\,\omega_j\in\Omega_j}{\mathbb{F}_i}\left[V_j(x_i',\omega_j)\right] \\ \text{s.t.} \quad & \bar{x}_i = x_i \\ & x_i' = T_i(\bar{x}_i,u_i,\omega_i) \\ & u_i \in U_i(\bar{x}_i,\omega_i),\end{aligned} \tag{3}$$

where the decision-rule $\pi_i(x_i,\omega_i)$ takes the value of u_i in the optimal solution.

Definition A *Decision-Hazard subproblem* is the optimization problem:

$$\begin{aligned}\mathbf{DH}_i(x_i):\quad V_i(x_i) = \min_{u_i,\bar{x}_i,x_{i,\omega_i}'} \quad & \underset{\omega_i\in\Omega_i;\,j\in i^+}{\mathbb{F}_i}\left[C_i(x_i,u_i,\omega_i) + V_j\left(x_{i,\omega_i}'\right)\right] \\ \text{s.t.} \quad & \bar{x}_i = x_i \\ & x_{i,\omega_i}' = T_i(\bar{x}_i,u_i,\omega_i), \quad \forall \omega_i \in \Omega_i \\ & u_i \in U_i(\bar{x}_i)\end{aligned} \tag{4}$$

where the decision-rule $\pi_i(x_i)$ takes the value of u_i in the optimal solution.

Note that with the appropriate modifications to the cost-to-go terms, a policy graph can contain both Hazard-Decision and Decision-Hazard nodes.

Finally, a few last pieces of terminology. Given a policy π, we can *simulate* the multistage stochastic programming problem by the procedure given in Algorithm 1. As a result, we end up with a sequence of nodes i^n, states x^n, controls u^n, noise terms ω^n, and costs c^n for $n \in \{1,\ldots,N\}$.

Algorithm 1: Simulating the policy.

```
set $x^1 = x_R$
set $n = 1$
set $i^0 = R$
while $i^{n-1^+} \neq \varnothing$ do
    sample $i^n$ from $i^{n-1^+}$
    sample $\omega^n$ from $\Omega_{i^n}$
    if $i^n$ is Hazard-Decision then
        set $u^n = \pi_{i^n}(x^n, \omega^n)$
    else
        set $u^n = \pi_{i^n}(x^n)$
    end
    set $c^n = C_{i^n}(x^n, u^n, \omega^n)$
    set $x^{n+1} = T_{i^n}(x^n, u^n, \omega^n)$
    set $n = n + 1$
end
```

We also need to define a *scenario* and the *cumulative cost* of a scenario.

Definition A *scenario* is a sequence of node and noise realizations

$$(i^1, \omega^1), (i^2, \omega^2), \ldots, (i^N, \omega^N).$$

Definition The *cumulative cost* of a scenario is $\sum_{n=1}^{N} c^n$.

4 | PROPOSED ALGORITHM

So far in this paper, we have defined a general class of multistage stochastic programming problems. Now, we describe an extension of the stochastic dual dynamic programming (SDDP) algorithm (Pereira and Pinto, 1991; Philpott and Guan, 2008; Shapiro, 2011; Girardeau et al., 2015; Guigues, 2016) to the policy graph setting. We note that SDDP is not the only algorithm for solving multistage stochastic programs. Other methods, such as progressive hedging (Rockafellar and Wets, 1991), decompose the model by *scenario*. We use SDDP because it naturally arises from the decomposition structure of a policy graph.

Our algorithm cannot solve a general policy graph. Instead, given a policy graph $\mathcal{G} = (R, \mathcal{V}, \mathcal{E}, \Phi)$, it requires the following assumptions.

(A1) The number of nodes in $\mathcal{V}$ is finite.

(A2) Every node in the policy graph is *hazard-decision*.

(A3) The sample space Ω_i of random noise outcomes is finite at each node $i \in \mathcal{N}$.

(A4) Given fixed ω_i and x_i, and excluding the risk-adjusted expectation term $\mathbb{F}_i[\cdot]$, the subproblem associated with each node $i \in \mathcal{N}$ can be formulated as a linear programming problem.

(A5) For every node $i \in \mathcal{N}$, there exists a bounded and feasible optimal control u_i for every achieved incoming state x_i and realization of the noise ω_i.

(A6) For every node $i \in \mathcal{N}$, the sub-graph rooted at node i has a positive probability of reaching a leaf node.

(A7) The outgoing state variable x_i' belongs to a polytope X_i.

At first glance, these assumptions seem restrictive. However, let us make some comments regarding them. Assumptions (A1) – (A5) are variations of standard assumptions in the literature (Philpott and Guan, 2008). Readers should note that, although unstated, the formulation as a policy graph induces the commonly required assumption that ω is a *stagewise-independent* random variable and that the nodal transitions are Markovian. Although, as we mentioned in Section 3.1, stagewise-dependent processes can be modelled with an appropriate state-space expansion. Moreover, as we showed in Figure 3, assumption (A2) is less taxing than first assumed since any decision-hazard node can be reformulated into two hazard-decision nodes with an expanded state-space (Street et al., 2018). Under certain technical assumptions, such as an appropriate constraint qualification, assumption (A4) can be relaxed to a convex optimization problem; see Girardeau et al. (2015) for details. Regardless, (A4) places a number of conditions (e.g., convexity and continuity) on the function C, mapping T and feasibility set U. Assumption (A6) enforces a discounted infinite-horizon view of the world as opposed to an expected long-run average cost view of the world. It is also another way of saying that the discount factor around a cycle cannot be 1. (In the limit as the discount factor approaches 1, the discounted infinite-horizon policy converges to the average cost policy (Bertsekas, 2005).) Finally, assumption (A7) restricts the state variables to a bounded set so that we cannot have a sequence of divergent state variables as we simulate a forward path via Algorithm 1.

There are many examples in the literature of similar algorithms and convergence proofs for various special cases of the policy graph. For example, in rough chronological order:

- Pereira and Pinto (1991) introduced the original SDDP algorithm for the case of a linear policy graph and a linear subproblem. However, the idea can be traced back to Benders decomposition (Benders, 1962) and the L-shaped method of Van Slyke and Wets (1969) (originally for two-stage problems, it was extended to the multistage case by Louveaux (1980) and Birge (1985)). SDDP has been studied extensively in the literature (Philpott and Guan, 2008; Shapiro, 2011; Shapiro et al., 2013; Girardeau et al., 2015; Guigues, 2016). We can also recommend the accessible introductions to SDDP provided by Newham (2008, Ch. 4), Guan (2008, Ch. 5), and Dowson (2018, Ch. 2).
- Many variants of the SDDP algorithm were presented in the literature, including AND (Donohue and Birge, 2006), CUPPS (Chen and Powell, 1999), and ReSa (Hindsberger, 2014).
- Philpott and Guan (2008) proved the almost sure convergence of a modified version of SDDP. Earlier proofs, such as those by Chen and Powell (1999) and Linowsky and Philpott (2005), made use of an unstated assumption regarding the application of the second Borel-Cantelli lemma (Grimmett and Stirzaker, 1992). To differentiate their modified algorithm from the original SDDP, Philpott and Guan refer to their algorithm as DOASA.
- Various authors (Gjelsvik et al. (1999); Philpott and de Matos (2012)) presented algorithms for the Markovian policy graph setting, although none provided convergence results.
- Girardeau et al. (2015) proved the almost sure convergence of DOASA-type algorithms on a linear policy graph and a convex subproblem.
- Guigues (2016) proved the almost sure convergence of DOASA-type algorithms on problems with coherent risk measures.
- Rebennack (2016) presented an algorithm and convergence result for a general acyclic policy graph.
- Nannicini et al. (2017) presented an algorithm and convergence result for a linear policy graph that contains one cycle.
- Warrington et al. (2019) presented an algorithm and convergence result for a cyclic policy graph with a single node

and no noise terms.

- Baucke et al. (2018) presented a very similar result to Warrington et al. (2019), but used an exact upper bounding function to extend the analysis to a cyclic policy graph with any finite number of nodes, but no stagewise-independent noise terms. However, note that any node with stagewise-independent noise terms can be expanded into a collection of nodes with one node for each discrete realization of the noise term. Thus it is always possible to transform a problem from the setting used in this paper into the setting of Baucke.
- Guigues (2018) presented an algorithm and convergence result for problems with a random number of stages; these can be viewed as a linear policy graph with additional arcs between nodes such that the graph remains acyclic.
- Shapiro and Ding (2019) present an algorithm (but no convergence result) for Markovian policy graphs with a single cycle. Their algorithm is a sub-class of the algorithm of Nannicini et al. (2017) because it only considers roll-outs of a fixed length. However, they show the computational and policy benefit of considering infinite horizon models in a real-world case study of the Brazilian Interconnected Power System.

The extension to SDDP we describe in the remainder of this section generalizes the body of work just outlined. The key difference in our work compared to the Markov decision process literature (Puterman, 1994) and the stochastic optimal control literature (Bertsekas, 2005) is that we consider continuous state and control variables. For readers unfamiliar with the SDDP algorithm, it can be viewed as a form of value iteration (Howard, 1960), approximate linear programming (de Farias and van Roy, 2003), or approximate dynamic programming (Powell, 2011) in which we exploit the structure of the problem through linear programming duality to derive provably optimal basis functions.

Due to assumptions (A1)–(A7), the risk-adjusted cost-to-go function $\mathbb{F}\left[V_j(x_i', \omega_j)\right]$ is convex with respect to the state variable x_i'. Therefore, it can be replaced by a variable θ_i and approximated by the maximum of a set of affine functions, which we refer to as *cuts*. Like SDDP, our algorithm constructs the set of cuts iteratively. Each iteration consists of two phases: a forward pass, which samples a sequence of subproblems and values for the state variables $\mathcal{S} = [(i_1, x_{i_1}'), (i_2, x_{i_2}'), \ldots]$; and a backward pass, which refines the approximation of the cost-to-go function by adding a new cut to each of the subproblems visited in the forward pass. The resulting approximated subproblem at node i after K iterations can be expressed as:

$$\begin{aligned}
\mathbf{SP}_i^K: \quad V_i^K(x_i, \omega_i) = \min_{u_i, \bar{x}_i, x_i', \theta_i} \quad & C_i(\bar{x}_i, u_i, \omega_i) + \theta_i \\
\text{s.t.} \quad & \bar{x}_i = x_i, \qquad [\lambda_i] \\
& x_i' = T_i(\bar{x}_i, u_i, \omega_i) \\
& u_i \in U_i(\bar{x}_i, \omega_i) \\
& x_i' \in X_i \\
& \theta_i \geq \alpha_i^k + \langle \beta_i^k, x_i' \rangle, \quad k \in \{1, 2, \ldots, K\},
\end{aligned} \tag{5}$$

where $\langle x, y \rangle = x^\top y$. Note that λ_i is the vector of dual variables associated with the constraints $\bar{x}_i = x_i$. In other words, λ_i is a valid subgradient for the function $V_i^K(x_i, \omega_i)$ with respect to x_i.

In the literature many authors (Pereira and Pinto, 1991; Philpott and Guan, 2008) compute the subgradient of V_i^K with respect to the incoming state variables x_i by computing a transformation of the dual of the transition constraints (i.e., $x_i' = T_i(\bar{x}_i, u_i, \omega_i)$), taking into account the feasible set of actions U_i. This approach is overly complicated. Our solution (also used by Girardeau et al. (2015)) is to make $\bar{x}_i$ a dummy variable with the constraint[2]:

$$\bar{x}_i = x_i, \quad [\lambda_i].$$

This has downside of adding one extra variable and constraint for each state variable, but results in a simpler subgradient calculation.

4.1 | Computing cut coefficients

We now explain how to calculate the cut coefficients α and β. Consider a node j, given an incoming state variable $\hat{x}$ and a realization of the noise ω_j. We can solve $\mathbf{SP}_j^K$ and record the optimal objective value $V_j^K(\hat{x}, \omega_j)$, which we denote $\bar{V}_{j,\omega_j}^K$, and the optimal value of the dual variable λ_j, which we denote $\bar{\lambda}_{j,\omega_j}^K$. Since V_j^K is convex with respect to x_j, we have that:

$$V_j^K(x_j, \omega_j) \geq \bar{V}_{j,\omega_j}^K + \langle \bar{\lambda}_{j,\omega_j}^K, x_j - \hat{x} \rangle,$$

Re-arranging terms we get:

$$V_j^K(x_j, \omega_j) \geq \left(\bar{V}_{j,\omega_j}^K - \langle \bar{\lambda}_{j,\omega_j}^K, \hat{x} \rangle \right) + \langle \bar{\lambda}_{j,\omega_j}^K, x_j \rangle.$$

However, recall that we wish to approximate the cost-to-go function :

$$\underset{j \in i^+;\, \omega_j \in \Omega_j}{\mathbb{F}} \left[V_j(x_i', \omega_j) \right].$$

To do so, consider the following proposition from Philpott et al. (2013, Proposition 4), which follows from *Danskin's* theorem (Shapiro et al., 2009, Theorem. 7.25), and which we re-state using our notation. (Recall that ξ is the *changed* probability distribution from Section 3.3.)

Proposition 1 *Suppose, for each $\omega \in \Omega$, that $\lambda(\hat{x}, \omega)$ is a subgradient of $V(x, \omega)$ at $\hat{x}$. Then, given ξ such that $\mathbb{F}[V(\hat{x}, \omega)] = \mathbb{E}_\xi[V(\hat{x}, \omega)]$, $\mathbb{E}_\xi[\lambda(\hat{x}, \omega)]$ is a subgradient of $\mathbb{F}[V(x, \omega)]$ at $\hat{x}$.*

Using this result, we can construct a valid cut via the method given in Algorithm 2.

```
Algorithm 2: Cut calculation algorithm.
    Given $\hat{x}$
    Solve $\mathbf{SP}_j^K(\hat{x}, \omega_j)$ for all $j \in i^+$ and $\omega_j \in \Omega_j$
    Compute $\xi$ so that $\mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] = \underset{j \in i^+; \omega_j \in \Omega_j}{\mathbb{F}}\left[\bar{V}_{j,\omega_j}^K\right]$
    Set $\beta_i^{K+1} = \mathbb{E}_\xi\left[\bar{\lambda}_{j,\omega_j}^K\right]$
    Set $\alpha_i^{K+1} = \mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] - \langle \beta_i^{K+1}, \hat{x} \rangle$
    Obtain the inequality $\mathbb{F}_{j \in i^+;\, \omega_j \in \Omega_j}\left[V_j(x_i', \omega_j)\right] = \theta_i \geq \alpha_i^{K+1} + \langle \beta_i^{K+1}, x_i' \rangle$
```

[2] Alexandre Street and Davi Valladão call this the *fishing* dual.

4.2 | An iteration

We now describe an iteration of our algorithm in more detail. Each iteration consists of two phases: a forward pass, which samples a sequence of nodes and points in the state-space; and a backward pass, which uses the cut calculation algorithm to refine the approximation of the cost-to-go function at the points in the state-space visited on the forward pass.

In the forward pass of our algorithm, we sequentially solve a sequence of subproblems. After solving the subproblem at each node i, we terminate the forward pass and begin the backward pass with probability $1 - \sum_{j \in i^+} \phi_{ij}$. (Recall, ϕ_{ij} is the probability of transitioning from node i to node j.) This probabilistic termination is equivalent to reaching a dummy leaf node with $C_i(\cdot, \cdot, \cdot) = 0$. If we do not terminate the forward pass, we sample a new child node and continue the forward pass. Note also that if a node has no children (i.e., $i^+ = \varnothing$), then $\sum_{j \in i^+} \phi_{ij} = 0$ and we terminate the forward pass with probability 1. In addition, we choose a large finite τ and terminate the forward pass once we have sampled τ nodes in the sequence. We refer to this τ as the *maximum depth*. We discuss the choice of τ in more detail in Section 4.6. Pseudo-code for the forward pass is given in Algorithm 3. Note that $rand()$ samples a uniformly distributed random variable in $[0, 1]$.

Algorithm 3: Forward pass.

```
choose large, finite $\tau$
set $x = x_R$
set $\mathcal{S} = [\ ]$
set $i = R$
while $(rand() > 1 - \sum_{j \in i^+} \phi_{ij}) \wedge (|\mathcal{S}| < \tau)$ do
    sample new $i$ from $i^+$ according to the transition matrix $\Phi$
    sample $\omega_i$ from $\Omega_i$
    solve $\mathbf{SP}_j^K(x, \omega_i)$
    append $(i, x_i')$ to the list $\mathcal{S}$
    set $x = x_i'$
end
```

Given a list of nodes visited on the forward pass (and corresponding sampled points in the state-space), the backward pass of our algorithm is identical to standard SDDP implementations. Pseudo-code for the backward pass is given in Algorithm 4. Note that because we may end the forward pass at a leaf node or due to a cycle, we need to check that a node has children before attempting to update the value function. This condition ($i^+ = \varnothing$) will only hold for the last element in $\mathcal{S}$ if the forward pass reached a leaf node. However, placing this check in the backward pass simplifies the logic in other parts of the algorithm. $reverse(\mathcal{S})$ loops through the elements in the list $\mathcal{S}$ in the reverse order that they were inserted, i.e., from last to first.

It is also important to note that one backward pass may add multiple cuts to the same node if the forward pass sampled that node multiple times before termination. Thus, there may be different numbers of cuts at different nodes in the graph. For notational simplicity, we shall continue to use V_i^K to mean the approximated subproblem with K cuts, and we assume that this always refers to the maximum number of cuts added to that subproblem.

Readers should also note that there is no mention of *cut sharing* in our algorithm. (Cut sharing is the practice of using one cut to approximate the cost-to-go function at different nodes in a scenario tree, effectively "sharing" the cut between them. For more, see Infanger and Morton (1996); Rebennack (2016).) This is because a policy graph can be thought of compressed scenario tree, where nodes in the scenario tree have been merged into a single policy graph node *if and only if the cuts could be shared between them in SDDP*. This suggests that SDDP is a natural solution algorithm for multistage stochastic programs formulated as a policy graph.

Algorithm 4: Backward pass.

```
given $\mathcal{S}$ from forward pass
for $(i, \hat{x})$ in $reverse(\mathcal{S})$ do
    if $i^+ = \varnothing$ then
        /* This check accounts for the case when the forward pass reached a leaf node.    */
        continue
    end
    for $j \in i^+$ do
        for $\omega_j \in \Omega_j$ do
            solve $\mathbf{SP}_j^K(\hat{x}, \omega_j)$
            set $\bar{\theta}_{j,\omega_j}^K$ to the optimal objective value
            set $\bar{\lambda}_{j,\omega_j}^K$ to the value of $\lambda_j$ in the optimal solution
        end
    end
    compute $\xi$ so that $\mathbb{E}_\xi\left[\bar{\theta}_{j,\omega_j}^K\right] = \mathbb{F}_{i_{j \in i^+; \omega_j \in \Omega_j}}\left[\bar{\theta}_{j,\omega_j}^K\right]$
    set $\beta_i^{K+1} = \mathbb{E}_\xi\left[\bar{\lambda}_{j,\omega_j}^K\right]$
    set $\alpha_i^{K+1} = \mathbb{E}_\xi\left[\bar{\theta}_{j,\omega_j}^K\right] - \langle \beta_i^{K+1}, \hat{x} \rangle$
    add the cut $\theta_i \geq \alpha_i^{K+1} + \langle \beta_i^{K+1}, x_i' \rangle$ to $\mathbf{SP}_i^K$
end
```

4.3 | Lower bound

Since the cuts at each node form an outer-approximation of the cost-to-go function, we can obtain a valid lower bound $\underline{y}$ to the multistage stochastic programming problem 2 by evaluating:

$$\underline{y} = \underset{i \in R^+;\, \omega_i \in \Omega_i}{\mathbb{F}} \left[V_i^K(x_R, \omega_i)\right].$$

4.4 | Upper bound

We do not have an exact upper bound for this algorithm. This is not unique to our algorithm and is common in SDDP-type algorithms. (Some recent work has been done on upper bounds in the linear policy graph case, see e.g., Baucke et al. (2018); Leclère et al. (2018).) In the case when all of the risk measures in the policy graph are the expectation operator, an unbiased estimate for the upper bound for the problem can be obtained by performing a Monte Carlo simulation of the policy. This approach is commonly used in SDDP with acyclic policy graphs (see, e.g., Pereira and Pinto (1991)). Pseudo-code is given in Algorithm 5. In the risk-averse case, we do not have an upper bound.

Algorithm 5: Upper bound calculation.

```
choose large, finite $\tau$
for $n$ = 1,…,$N$ do
    set $x = x_R$
    set $i = R$
    set $y_n = 0$
    while $(rand() > 1 - \sum_{j \in i^+} \phi_{ij}) \ \wedge (|\mathcal{S}| < \tau)$ do
        sample new $i$ from $i^+$ according to the transition matrix $\Phi$
        sample $\omega_i$ from $\Omega_i$
        solve $\mathbf{SP}_j^K(x, \omega_i)$
        set $y_n = y_n + V_i^K(x, \omega_i)$
        set $x = x_i'$
    end
end
Calculate the sample mean
$\bar{y} = \frac{1}{N} \sum_{n=1}^{N} y_n$
Form a confidence interval around $\bar{y}$ for the population mean based on $\{y_n\}_1^N$
```

4.5 | Termination

The upper bound discussed above is statistical. Recently, some work has explored deterministic upper bounds (Philpott et al., 2013; Baucke et al., 2018; Leclère et al., 2018), but we have not attempted to adapt those results to our setting. In lieu of a deterministic upper bound, the question of when to terminate SDDP-type algorithms is an open question. Various rules for terminating have been proposed in the literature, and all have their strengths and weaknesses. The most commonly used termination criteria is to terminate the algorithm once the lower bound is within the confidence interval of the upper bound (Pereira and Pinto, 1991). We direct the reader to Homem-de-Mello et al. (2011) for a discussion on some of the subtler aspects of this criteria.

4.6 | Convergence argument

Let us turn our attention to the convergence of our algorithm. In many cases, SDDP-type algorithms have been shown to converge to an optimal policy almost surely in a finite number of iterations (Philpott and Guan, 2008; Girardeau et al., 2015; Guigues, 2016). We do not claim such a result for our algorithm on a general policy graph. In part, this is because duality results in the infinite-horizon setting are often challenging to obtain (Ghate and Smith, 2013). However, the large finite τ in our algorithm induces a truncation of the full infinite-horizon problem into a finite-horizon problem. To understand this, consider the cyclic policy graph given in Figure 15.

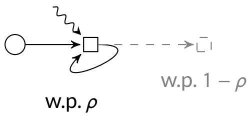

FIGURE 15 Cyclic policy graph with implicit leaf node.

If we choose the maximum depth to be $\tau = 4$, then we can *un-roll* the cycle of the policy graph to obtain the equivalent policy graph given in Figure 16. Note that we have slightly abused our notation in Figure 16 since the decision-rule at each node is identical and all four hazard-decision nodes will share the same set of cuts approximating the cost-to-go function.

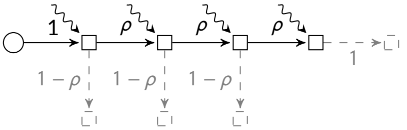

FIGURE 16 Unrolled cyclic policy graph with implicit leaf nodes in the case where $\tau = 4$.

Importantly, the unrolled policy graph is no longer cyclic. Instead, it is an acyclic Markovian policy graph. Therefore, given the truncation, the finite convergence of our algorithm follows directly from existing proofs in the literature (Philpott and Guan, 2008; Girardeau et al., 2015; Guigues, 2016). However, because of the truncation, we do not obtain the optimal policy for the original problem. Moreover, because each node shares the same set of cuts, we do not obtain the optimal policy to the linear policy graph with τ stages. Instead, assuming that the truncated problem is well-behaved compared to the original problem (in the sense that as τ increases, the optimal policy of the truncated problem converges to the optimal policy of the original problem), it is possible to bound the truncation error by some value ε as a function of the maximum depth τ. The relationship between ε and τ depends on the structure of the policy graph. Thus, instead of presenting a general result, we will prove the finite $\epsilon-$convergence of our algorithm for a simple cyclic policy graph containing one node that loops back onto itself (Figure 15). Despite the simple setting, readers should easily see how the proof can be extended to more complicated policy graphs.

To begin, assume that there exists a large, finite M such that $|C_i(x_i, u_i^*, \omega_i)| \leq M < \infty$ for all $i \in \mathcal{N}$, $\omega_i \in \Omega_i$, and achievable values for the incoming state x_i. A value for M always exists since by assumptions (A4), (A5), and (A7), each subproblem is a bounded and feasible linear program, and the set of possible values for x_i is also bounded. Then

$$\frac{-M}{1-\rho} \leq \mathop{\mathbb{F}}_{j\in i^+;\, \omega_j\in\Omega_j}\left[V(x_i', \omega_j)\right] \leq \frac{M}{1-\rho}, \qquad (6)$$

where u_i^* is the optimal control.

By assumption (A6), $\rho < 1$, and so $M/(1-\rho)$ is finite. Now consider unrolling the cycle of the policy graph in Figure 15 so that it contains τ nodes. As we have already seen, an example where $\tau = 4$ is given in Figure 16. Based on the bound derived in (6), we know that in the last node, a bound on the cost-to-go that we have not modeled (because we truncated the infinite-horizon problem to τ nodes) is:

$$\left|\mathop{\mathbb{F}}_{i\in R^+;\, \omega_i\in\Omega_i}\left[V(x_R, \omega_i)\right] - \mathop{\mathbb{F}}_{i\in R^+;\, \omega_i\in\Omega_i}\left[\bar{V}^*(x_R, \omega_i)\right]\right| = \epsilon \leq \rho^\tau \times \frac{2M}{1-\rho},$$

where $\bar{V}^*$ is the converged cost-to-go function of the truncated problem. This error ϵ can be made arbitrarily small by choosing a large, finite τ.

We note that our algorithm, and this convergence argument, is very similar to the work of Nannicini et al. (2017). The key difference is that we choose a maximum depth τ *a priori* and use the transition probabilities to terminate the forward pass, whereas Nannicini et al. (2017) use an iterative method to slowly increment the value of τ until the required ϵ-convergence is achieved, and they do not use the transition probabilities to terminate the forward pass.

Moreover, due to policy graph formulation, our algorithm applies to arbitrary graphs, as opposed to the linear policy graph with a single cycle considered by Nannicini et al. (2017). However, it is likely that their method also extends to the arbitrary policy graph case.

We also highlight the work of Legat and Jungers (2016) who (in an application arising from information theory and not stochastic programming) proposed an algorithm for a cyclic policy graph when $C_i(\cdot, \cdot, \cdot) = 0$. Like our algorithm, their algorithm includes a maximum depth limit that is used to terminate the forward pass. Notably, their work considers problems without relatively complete recourse, and so Benders feasibility cuts are generated at each node in the graph.

5 | EXAMPLE: PASTORAL DAIRY FARMING

To demonstrate the practicality of our algorithm, we solve a single instance of a multistage stochastic programming model from the literature. Since we only solve this one instance, we will not draw strong conclusions about the computational performance of the method, or the impact on solution times and policy quality of choosing different parameters (such as the discount factor). We leave these tasks for future work.

The POWDer model of Dowson et al. (2019) is a multistage stochastic programming model of a pastoral dairy farm over the course of one year. POWDer decomposes the year into a Markovian policy graph with 52 weekly stages. The decision process is modelled as follows. At the beginning of week $t = 1, 2, \ldots, 52$, the farmer measures the five state variables in the model: the soil moisture W_t (mm), the pasture cover P_t (kg/ha), the quantity of grass in storage Q_t (kg/ha), the number of cows milking C_t (cows/ha), and the quantity of milk solids produced to date M_t (kg/ha). Before choosing an action for the week, the farmer observes the realization of the two stagewise-independent random variables: the potential evapotranspiration[3] e_t^p (mm), and the quantity of rainfall r_t (mm). The Markov states in the Markovian policy graph correspond to a forecast for the price that the farmer receives for their milk in the last stage of each year (\$/kg). We refer to this price as the *end-of-season* milk price. During the year, we refer to the forecast for the end-of-season milk price associated each Markov state as the *forecast* milk price. Taking into account the incoming values of the state variables, the Markov state p_t, and the observation of the stagewise-independent noise term, the farmer decides the quantity of pasture to harvest h_t (kg/ha), the number of cows to stop milking u_t (cows/ha), and the quantities of grass from pasture f_t^p, grass from storage f_t^q, and palm kernel s_t to feed the herd (all kg/ha). As a result of these actions, the system transitions to a new state that will serve as the incoming state in the next week, and the farmer incurs the cost of purchasing palm kernel, harvesting pasture, and applying irrigation. In the last week, the forecast milk price becomes the end-of-season milk price, and the farmer sells all of the milk produced during the season M_{53} (kg/ha) at the end-of-season milk price p_{52} (\$/kg).

As a simple example of the dynamics in the model, consider Q_t (kg/ha), the quantity of grass in storage at the start of week t. This is a state variable with linear dynamics:

$$Q_{t+1} = Q_t + \beta h_t - f_t^q,$$

where h_t (kg/ha) is the quantity harvested, β is the harvesting efficiency < 1, and f_t^q (kg/ha) is the amount of grass from storage fed to the cows. As a result, the farmer incurs a cost of $c^h \times h_t$ in week t, where c^h is the cost of harvesting one kg of grass (a constant in the model).

POWDer is able to model the effect of weather and price uncertainty on the management actions of the pastoral dairy farmer. However, because of the finite-horizon assumption, the authors introduced a penalty if the pasture

[3] A function of sunlight and temperature that is positively correlated to grass growth.

cover at the end of the year was less than the pasture cover at the start of the season. In this section, we relax this constraint and model POWDer as an infinite-horizon problem. We retain the 52 weekly stages and Markovian policy graph structure used in Dowson et al. (2019), and assume that the end of week 52 is equivalent to the start of week 1. A graphical representation of the policy graph is given in Figure 17. We assume that the discount factor associated with the arcs exiting nodes in the 52$^{\text{nd}}$ stage is <1.

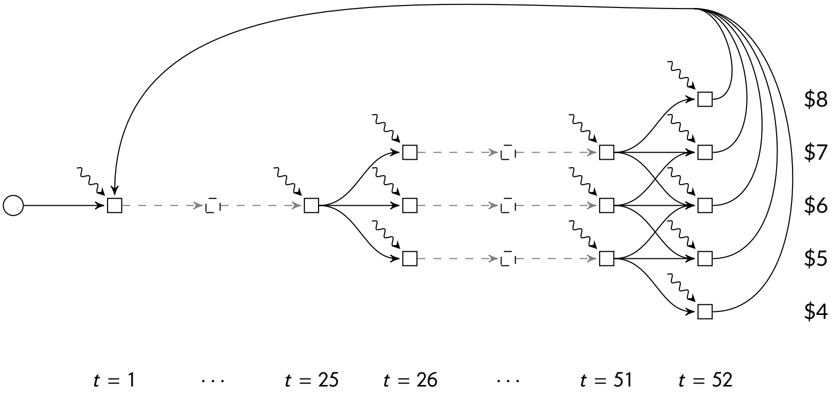

FIGURE 17 Infinite-horizon Markovian policy graph of the POWDer model. Stages correspond to weeks of the year, Markov states correspond to forecasts for the end-of-season milk price.

5.1 | Results

Based on our domain knowledge of the POWDer model, we can derive an lower bound for the expected value of each season by assuming that no milk is produced and all of the energy required by the cows is derived from palm kernel. Using the data from Dowson et al. (2019), a conservative value for this lower bound is –$11,000/ha/year. In addition, we can derive an upper bound for the expected value of each season by assuming that every cow produces the maximum quantity of milk, that the farmer sells this milk at the expected end-of-season milk price, and that they incur no costs. Using the data from Dowson et al. (2019), a conservative value for this upper bound is +$10,000/ha/year. Thus, we can obtain a bound on ϵ as a function of the discount factor ρ and maximum depth τ of:

$$\epsilon \leq \rho^{\tau/52} \times \frac{2 \cdot 11,000}{1 - \rho}. \tag{7}$$

We solved the infinite-horizon POWDer model for 1000 iterations using a discount factor of $\rho = 0.95$ and maximum depth of $\tau = 18,200$ (350 cycles). Thus, from (7), we know that $\epsilon \leq \$0.01/\text{ha}$. Also note that the probability of reaching τ in the forward pass is $\approx 10^{-8}$ (i.e., 0.95^{350}). This value is so small that in practice we never complete a forward pass (in either the training iterations or the simulation of the policy) that reaches stage τ. After iterations 200, 400, 600, 800, and 1000, we performed a Monte Carlo simulation of the policy with 1000 replications to obtain a confidence interval for the lower bound. (Note that upper and lower are reversed since POWDer is a maximization model.) A plot of the lower and upper bounds as a function of the number of iterations is given in Figure 18.

We also simulated the policy after 1000 iterations with a Monte Carlo simulation of 1000 replications. Unlike the

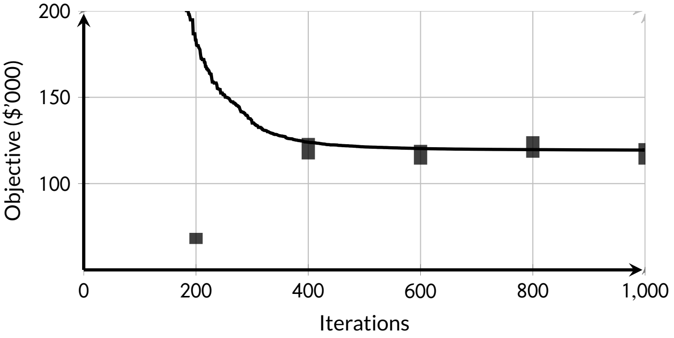

FIGURE 18 Convergence of the upper bound against the number of iterations for the POWDer model with a discount factor of $\rho = 0.95$. Thick vertical bars are 95% confidence intervals for the mean of the Monte Carlo simulation of the policy. The upper bound in the first 200 iterations is greater than $200,000.

simulation used to obtain an estimate for the lower bound, we did not terminate the forward pass with probability $1 - \rho$ at the end of each season. Instead, we terminated the forward pass once five complete seasons had been simulated. This simulation is visualized in Figure 19. In each of the subplots, we plot the 0–100 percentiles of the distribution of the plotted variable as a light shaded band. The dark shaded bands correspond to the 10–90th percentiles. The dotted line corresponds to the 50th percentile. Figure 19a shows the number of cows that are milking (a state variable) at the end of each stage. We start each season (August 1st) will all cows milking (i.e., 3 cows/ha). At some point during the season, the farmer stops milking the cows, and the number of cows milking drops to zero. In Figure 19b we plot the pasture cover (a state variable) at the end of stage. Excluding the first season, the pasture cover exhibits strong seasonality with a minimum around 500 kg/ha in July and a maximum of 2000 kg/ha in December. This seasonality is driven by the evapotranspiration (a stagewise-independent noise), which we plot in Figure 19c. Since evapotranspiration is a function of the incoming solar radiation and air temperature, it has a maximum in the New Zealand summer (Dec - Feb) and a minimum in the New Zealand winter (Jun - Aug).

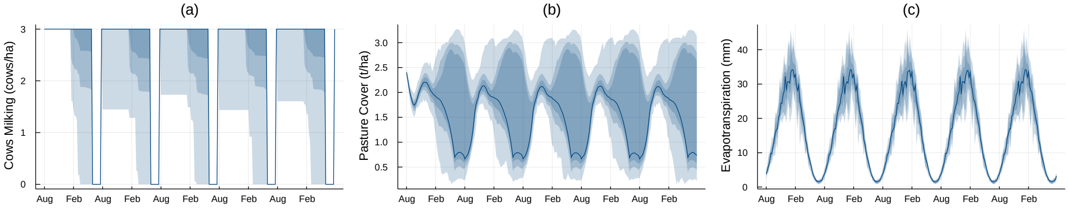

FIGURE 19 Simulation of the infinite-horizon policy over five seasons.

In Dowson et al. (2019), the authors placed a large penalty on ending the season with a pasture cover of less than 2,500 kg/ha. They justified this by arguing that it was necessary for the pasture cover at the end of the year to be no less than the pasture cover at the beginning of the year in order for the same policy to be used in consecutive years. However, as shown in Figure 19b, this constraint is sub-optimal. Instead of ending the year with a pasture cover of 2,500 kg/ha (above the long-run maximum), the optimal solution in the infinite-horizon policy is to end the season with the minimum pasture cover, enabling the pasture cover to synchronize with the evapotranspiration. We can also compare the discounted expected value the finite-horizon and infinite-horizon policies. In the finite-horizon case, POWDer converges to a value of \$5,782/ha/year. Discounting this value at a discount factor of $\rho = 0.95$ gives a discounted expected value of \$115,640/ha. In comparison, the infinite-horizon policy converges to a value of \$119,439/ha. Thus, the infinite-horizon policy leads to an increase of at least \$3,799/ha (+3.3%) in the discounted expected value. This demonstrates that the policy in the original paper of Dowson et al. (2019) was sub-optimal because of end-of-horizon effects.

Regardless of the increase in value, the main benefit of the infinite-horizon model is that practitioners no longer have to derive reasonable estimates for the terminal value function, or worry about the effect that their choice of terminal value function has on the resulting policy. However, the main draw-back of solving the infinite-horizon model is an increased computational cost. The difference in computational cost is qualitatively related to the discount factor ρ: as ρ increases, the computational cost increases because the discounting becomes relevant over longer and longer time horizons. The finite horizon problem is easier to solve because it corresponds to setting $\rho = 0$. (For more on this topic, see Shapiro and Ding (2019).) Ultimately, it is up to the practitioner to make an application-dependent choice between a finite-horizon and an infinite-horizon model, and to choose the discount factor within an infinite-horizon model.

6 | CONCLUSION

This paper introduced the *policy graph* as a structured way of formulating multistage stochastic programming problems. The policy graph representation unifies many of the existing special case formulations of multistage stochastic programming problems in the literature into one general framework. This paper also proposed an extension to the stochastic dual dynamic programming algorithm to the case of a general policy graph. By truncating the infinite-horizon problem with some small bounded error ϵ, we are able to recover an acyclic policy graph for which there are existing proofs of convergence.

Although this paper has presented one application from agriculture, there are many more applications that this could be applied to. Indeed, in many real-world applications, the natural model is an infinite-horizon formulation as opposed to a finite-horizon formulation.

SUPPLEMENTARY MATERIALS

To facilitate the adoption of our algorithm, we provide an implementation in the open-source multistage stochastic programming solver `SDDP.jl` (Dowson and Kapelevich, 2017). The online documentation for the solver (`https://odow.github.io/SDDP.jl/latest/index.html`) contains a number of tutorials to help new users model and solve multistage stochastic programming problems using `SDDP.jl`.

ACKNOWLEDGEMENTS

We thank David Morton, Tony Downward, Andy Philpott, Andrew Mason, Vincent Leclère, François Pacaud, Benoît Legat, and Joaquim Dias Garcia for the many useful discussions that directly led to this work. We also thank the anonymous reviewer for their helpful suggestions. b

REFERENCES

Artzner, P., Delbaen, F., Eber, J.-M. and Heath, D. (1999) Coherent measures of risk. *Mathematical Finance*, **9**, 203–228.

Baucke, R., Downward, A. and Zakeri, G. (2018) A deterministic algorithm for solving multistage stochastic minimax dynamic programmes. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2018/02/6449.html.

Bellman, R. (1954) The Theory of Dynamic Programming. *Bulletin of the American Mathematical Society*, **60**, 503–515.

Benders, J. (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, **4**, 238–252.

Bertsekas, D. (2005) *Dynamic Programming and Optimal Control*, vol. 1. Bellmont, MA: Athena Scientific, third edn.

Birge, J. R. (1985) Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs. *Operations Research*, **33**, 989–1007.

Chen, Z. L. and Powell, W. B. (1999) Convergent Cutting-Plane and Partial-Sampling Algorithm for Multistage Stochastic Linear Programs with Recourse. *Journal of Optimization Theory and Applications*, **102**, 497–524.

de Farias, D. and van Roy, B. (2003) The linear programming approach to approximate dynamic programming. *Operations Research*, **51**, 850–865.

Donohue, C. J. and Birge, J. R. (2006) The abridged nested decomposition method for multistage stochastic linear programs with relatively complete recourse. *Algorithmic Operations Research*, **1**.

Dowson, O. (2018) *Applying Stochastic Optimisation to the New Zealand Dairy Industry*. PhD Thesis, University of Auckland, Auckland, New Zealand.

Dowson, O. and Kapelevich, L. (2017) SDDP.jl: A Julia package for Stochastic Dual Dynamic Programming. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2017/12/6388.html.

Dowson, O., Philpott, A., Mason, A. and Downward, A. (2019) A mutistage stochastic optimization model of a pastoral dairy farm. *European Journal of Operational Research*, **274**, 1077–1089.

Ghate, A. and Smith, R. L. (2013) A Linear Programming Approach to Nonstationary Infinite-Horizon Markov Decision Processes. *Operations Research*, **61**, 413–425.

Girardeau, P., Leclère, V. and Philpott, A. B. (2015) On the Convergence of Decomposition Methods for Multistage Stochastic Convex Programs. *Mathematics of Operations Research*, **40**, 130–145.

Gjelsvik, A., Belsnes, M. and Haugstad, A. (1999) An algorithm for stochastic medium term hydro thermal scheduling under spot price uncertainty. In *PSCC: 13th Power Systems Computation Conference : Proceedings*, 1079–1085. Trondheim: Executive Board of the 13th Power Systems Computation Conference, 1999.

Grimmett, G. and Stirzaker, D. (1992) *Probability and Random Processes*. Oxford: Oxford University Press, second edn.

Guan, Z. (2008) *Strategic Inventory Models for International Dairy Commodity Markets*. PhD Thesis, University of Auckland, Auckland, New Zealand.

Guigues, V. (2014) SDDP for some interstage dependent risk-averse problems and application to hydro-thermal planning. *Computational Optimization and Applications*, **57**, 167–203.

— (2016) Convergence Analysis of Sampling-Based Decomposition Methods for Risk-Averse Multistage Stochastic Convex Programs. *SIAM Journal on Optimization*, **26**, 2468–2494.

— (2018) Multistage stochastic programs with a random number of stages: Dynamic programming equations, solution methods, and application to portfolio selection. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2018/03/6530.html.

Hindsberger, M. (2014) ReSa: A method for solving multistage stochastic linear programs. *Journal of Applied Operational Research*, **6**, 2–15.

Homem-de-Mello, T., de Matos, V. L. and Finardi, E. C. (2011) Sampling strategies and stopping criteria for stochastic dual dynamic programming: A case study in long-term hydrothermal scheduling. *Energy Systems*, **2**, 1–31.

Homem-de-Mello, T. and Pagnoncelli, B. K. (2016) Risk aversion in multistage stochastic programming: A modeling and algorithmic perspective. *European Journal of Operational Research*, **249**, 188–199.

Howard, R. (1960) *Dynamic Programming and Markov Processes*. Cambridge, MA.: MIT Press.

Infanger, G. and Morton, D. P. (1996) Cut sharing for multistage stochastic linear programs with interstage dependency. *Mathematical Programming*, **75**, 241–256.

Leclère, V., Carpentier, P., Chancelier, J.-P., Lenoir, A. and Pacaud, F. (2018) Exact converging bounds for Stochastic Dual Dynamic Programming via Fenchel duality. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2018/04/6575.html.

Legat, B. and Jungers, R. (2016) Parallel optimization on the Entropic Cone. In *Proceedings of the 37rd Symposium on Information Theory in the Benelux*, SITB '16, 206–211. Louvain-la-Neuve, Belgium.

Linowsky, K. and Philpott, A. B. (2005) On the Convergence of Sampling-Based Decomposition Algorithms for Multistage Stochastic Programs. *Journal of Optimization Theory and Applications*, **125**, 349–366.

Louveaux, F. (1980) A Solution Method for Multistage Stochastic Programs with Recourse with Application to an Energy Investment Problem. *Operations Research*, **28**, 889–902.

Nannicini, G., Traversi, E. and Calvo, R. W. (2017) A Benders squared (B2) framework for infinite-horizon stochastic linear programs. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2017/06/6101.html.

Newham, N. (2008) *Power System Investment Planning Using Stochastic Dual Dynamic Programming*. Ph.D. thesis, University of Canterbury, Christchurch, New Zealand.

NIWA (2018) El Niño and La Niña. https://www.niwa.co.nz/climate/information-and-resources/elnino. [Online; accessed 2018-04-03].

Pereira, M. and Pinto, L. (1991) Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, **52**, 359–375.

Pflug, G. C. and Pichler, A. (2016) Time-inconsistent multistage stochastic programs: Martingale bounds. *European Journal of Operational Research*, **249**, 155–163.

Philpott, A., de Matos, V. and Finardi, E. (2013) On Solving Multistage Stochastic Programs with Coherent Risk Measures. *Operations Research*, **61**, 957–970.

Philpott, A. B. and de Matos, V. L. (2012) Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion. *European Journal of Operational Research*, **218**, 470–483.

Philpott, A. B. and Guan, Z. (2008) On the convergence of sampling-based methods for multi-stage stochastic linear programs. *Operations Research Letters*, **36**, 450–455.

Powell, W. B. (2011) *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley Series in Probability and Statistics. Hoboken, N.J: Wiley, 2nd ed edn.

— (2014) Clearing the Jungle of Stochastic Optimization. In *Bridging Data and Decisions* (eds. A. M. Newman, J. Leung and J. C. Smith), TutORials in Operations Research, 109–137. INFORMS.

— (2016) A Unified Framework for Optimization under Uncertainty. In *Optimization Challenges in Complex, Networked and Risky Systems* (eds. A. Gupta, A. Capponi and J. C. Smith), TutORials in Operations Research, 45–83. INFORMS.

Puterman, M. (1994) *Markov Decision Processes*. Wiley Series in Probability and Statistics. New York, NY: John Wiley & Sons.

Rebennack, S. (2016) Combining sampling-based and scenario-based nested Benders decomposition methods: Application to stochastic dual dynamic programming. *Mathematical Programming*, **156**, 343–389.

Rockafellar, R. T. and Wets, R. J.-B. (1991) Scenarios and Policy Aggregation in Optimization Under Uncertainty. *Mathematics of Operations Research*, **16**, 119–147.

Rockafellar, T. R. and Uryasev, S. P. (2002) Conditional Value-at-Risk for General Loss Distributions. *Journal of Banking and Finance*, **26**, 1443–1471.

Ruszczyński, A. and Shapiro, A. (2006) Conditional Risk Mappings. *Mathematics of Operations Research*, **31**, 544–561.

Shapiro, A. (2011) Analysis of stochastic dual dynamic programming method. *European Journal of Operational Research*, **209**, 63–72.

Shapiro, A., Dentcheva, D. and Ruszczynski, A. (2009) *Lectures on Stochastic Programming: Modelling and Theory*. Philadelphia: Society for Industrial and Applied Mathematics.

Shapiro, A. and Ding, L. (2019) Stationary Multistage Programs. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2019/09/7367.html.

Shapiro, A., Tekaya, W., da Costa, J. P. and Soares, M. P. (2013) Risk neutral and risk averse Stochastic Dual Dynamic Programming method. *European Journal of Operational Research*, **224**, 375–391.

Sterman, J. (2000) *Business Dynamics: Systems Thinking and Modeling for a Complex World*. McGraw-Hill Higher Education.

Street, A., Lawson, A., Valladao, D. and Velloso, A. (2018) On the Solution of Decision-Hazard Multistage Stochastic Hydrothermal Scheduling Problems. *Optimization Online*. http://www.optimization-online.org/DB_HTML/2018/10/6855.html.

Valladão, D., Silva, T. and Poggi, M. (2019) Time-consistent risk-constrained dynamic portfolio optimization with transactional costs and time-dependent returns. *Annals of Operations Research*, **282**, 379–405.

Van Slyke, R. M. and Wets, R. (1969) L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *SIAM Journal on Applied Mathematics*, **17**, 638–663.

Warrington, J., Beuchat, P. N. and Lygeros, J. (2019) Generalized dual dynamic programming for infinite horizon problems in continuous state and action spaces. *IEEE Transactions on Automatic Control*, **64**, 5012–5023.