MDP modeling for multi-stage stochastic programs

David P. Morton[1], Oscar Dowson[2], and Bernardo K. Pagnoncelli[3]

[1]Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA; david.morton@northwestern.edu

[2]Dowson Farms, Auckland, New Zealand; oscar@dowsonfarms.co.nz

[3]SKEMA Business School, Université Côte d'Azur, Lille, France; bernardo.pagnoncelli@skema.edu

April 2026

Abstract

We study a class of multi-stage stochastic programs, which incorporate modeling features from Markov decision processes (MDPs). This class includes structured MDPs with continuous state and action spaces. We extend policy graphs to include decision-dependent uncertainty for one-step transition probabilities as well as a limited form of statistical learning. We focus on the expressiveness of our modeling approach, illustrating ideas with a series of examples of increasing complexity. As a solution method, we develop new variants of stochastic dual dynamic programming, including approximations to handle non-convexities.

Keywords: multi-stage stochastic programming; Markov decision processes; policy graph; decision-dependent uncertainty; statistical learning

1 Introduction

We study sequential decision problems under uncertainty, in which an agent takes an action at each time period that influences both the present (e.g., a reward is received) and the future (e.g., the state will change in the next time period). Multiple modeling approaches handle sequential decision-making under uncertainty; see, e.g., [44]. The two most relevant to our work are multi-stage stochastic programming (MSP) [9, 29, 48] and Markov decision processes (MDPs) [6, 12, 26, 43, 45]. From the perspective of MSP, we extend stochastic programs to incorporate attractive modeling features of MDPs, including a type of *decision-dependent uncertainty* and a form of *statistical learning*. These extensions can be solved by variants of stochastic dual dynamic programming (SDDP), a workhorse algorithm for large-scale MSP with many stages; see the seminal work of [41], as well as the surveys of [18] and [47, Ch. 2]. From the perspective of MDPs, we provide algorithms for solving a class of offline problems—which we call CACS MDPs—with continuous action and continuous state spaces, avoiding the need for discretization.

A discrete time, action, and state MDP has states $\mathcal{S}$, state-dependent actions $\mathcal{A}_s$, and a transition function, $\mathbb{P}(s'|s,a)$, which gives the one-step probability of moving to state s' conditioned on taking action a in state s. As the notation indicates, $\mathbb{P}(s'|s,a)$ depends on action a. The solution to an MDP is a policy, $\pi(a|s)$, that gives the (typically degenerate) probability of taking action a in state s, and that maximizes a function of the rewards from a starting state, which is governed by distribution $\mu(s)$.

Many authors have relaxed the classical assumption that an MDP's parameters are known. Wiesemann et al. [53] deal with uncertainty in one-step transition probabilities using robust optimization. Wang et al. [52] use covariates instead of process trajectories to deal with uncertain transition and reward functions. Reinforcement learning (RL) deals with sequential decision problems [4, 36] by simultaneously estimating (i.e., learning) an optimal policy and the model's transition and reward functions.

When the action and state spaces are finite and of modest size, exact dynamic programming algorithms can solve MDPs. Backward induction is used in finite-horizon problems, and in infinite-horizon problems policy iteration, value iteration, and their variants are used [45]. As the size of the spaces grows, the cost-to-go

functions are approximated, either using sampling [11], assuming a specific functional form (e.g., separable approximations [51]), using basis functions (e.g., [39]), or via deep learning [34]. While there are exceptions (e.g., [31]), most work in computational MDPs uses discrete actions and states.

We extend MSP with MDP modeling constructs, incorporating decision-dependent one-step transition probabilities and learning from a set of hypothesized models. Doing so frames stochastic programming as a *modeling* and *solution technique* for CACS MDPs. Using *policy graphs* [13], i.e., stochastic programs defined on Markov chains, we show that a broad class of MDPs can be modeled via MSP and approximately solved using SDDP. An SDDP algorithm exploits convexity and duality to construct a polyhedral outer approximation of the cost-to-go functions. When states and actions are continuous—and in some restricted cases, discrete—and the transition and cost functions are convex, that approximation is asymptotically optimal [42, 20, 23, 13]. Because SDDP uses convex optimization, the algorithm can find policies for problems with thousands of continuous control variables and a complicated, but still convex, feasible region.

Our decision-dependent one-step transition probabilities introduce non-convexities. The convexity assumptions of SDDP have been interpreted as precluding its application to non-convex problems. However, a recent trend computes a convex value function approximation to define a sub-optimal policy, which is then evaluated using the original non-convex model. Our work is part of this trend. When non-convexities arise due to discrete variables, Lagrangian duality can construct a convex approximation [55]. This technique has been applied, for example, to infrastructure planning problems in electric power systems [32, 33]. In hydrothermal scheduling, Rosemberg et al. [46] build a value function using convex approximations of the alternating current optimal power flow (AC-OPF) problem, and then evaluate the resulting policy using non-convex AC-OPF dynamics. In a problem with a hidden Markov model, Siddig et al. [49] convexify the value function by assuming a fully observable Markov chain, but then evaluate the policy on the hidden model.

While inherent in MDPs, decision-dependent uncertainty, in which the agent's actions change the probability distribution, is less common in stochastic programs. That said, there is a stream of stochastic programming literature on this topic, and we point to [1, Chapter 5] and [25] for reviews. In one class of problems, the agent's decisions alter the probability mass function of a fixed set of realizations according to a decision-dependent function [3, 35, 54]. In a second class of problems, the points of support, or in time-dynamic problems the information structure, is altered by the agent's actions [5, 21, 30]. In isolation, our decision-dependent approach is of the former type, although we alter one-step transition probabilities in a Markov chain rather than the probability mass of a random vector, and our approach applies to multi-stage problems with an infinite horizon. When combined with statistical learning, our approach can be of the second type in that the agent's actions can alter the resulting points of support.

The paper focuses on demonstrating the flexibility and expressiveness of our approach to modeling—primarily through several examples—although we derive requisite new SDDP algorithms. The contributions of our paper are:

1. We extend policy graphs and SDDP to allow the one-step transition probabilities to depend on our decisions.

2. We incorporate learning so an agent can probe and gather information to improve their beliefs before committing to a decision or, more generally, while committing to a sequence of decisions.

From one lens, these contributions add MDP- and RL-style modeling features to MSPs. From another, our SDDP algorithms can approximately solve a class of CACS MDPs. To summarize, we extend SDDP to handle a class of stochastic programs defined on Markov chains that incorporate decision-dependent uncertainty and learning, enabling continuous-state MDP modeling without discretization.

The paper is laid out as follows. Section 2 introduces policy graphs, which we extend to decision-dependent transitions, to a form of statistical learning, and to decision-dependent learning in Sections 3, 4, and 5. Section 6.1 summarizes a basic SDDP algorithm, which we extend to decision-dependent learning in Sections 6.2 and 6.3. Section 7 presents numerical experiments.

2 Policy graphs for MDPs

With an eye towards extensions, we present—with some necessary changes to the notation and definitions—the *policy graph* modeling framework of [13]. We use x for the physical state variable and u for the control variable, consistent with the stochastic control literature. The goal in this section is to establish the core modeling approach and then extend its reach in subsequent sections.

2.1 Mathematical model

Definition 1. A *policy graph*, $\mathcal{G} = (R, \mathcal{N}, \Phi)$, is composed of a tuple with a root node R, a further set of nodes $\mathcal{N}$, and an $|\mathcal{N}| + 1$ by $|\mathcal{N}|$ matrix Φ that specifies one-step transition probabilities, with entries ϕ_{ij}. We use Φ_0 to denote the square matrix of transition probabilities on $\mathcal{N}$, dropping the root. The *children* of node i are $i^+ = \{j \in \mathcal{N} : \phi_{ij} > 0\}$. A policy graph specifies a discrete-time, time-homogeneous, absorbing Markov chain with initial state specified by $\phi_{R,i}$, transition probabilities among transient states given by Φ_0, and a single absorbing state. For each i, $1 - \sum_{j \in i^+} \phi_{ij} \geq 0$ is the one-step probability of transitioning to the absorbing state, which has cost zero and ends a sequence of transitions. We do not explicitly model the absorbing state.

A policy graph includes a *decision rule*, $\pi_i(x, \omega_i)$, for each node that maps the *incoming physical state variable* x and realization of a *random variable* ω_i to a feasible *control variable* u and an *outgoing physical state variable* x' for a cost of $C_i(u, x', \omega_i)$, where $(u, x') = \pi_i(x, \omega_i) \in \mathcal{X}_i(x, \omega_i) \equiv \{(u, x') : u \in U_i(x, \omega_i),\ x' = T_i(u, x, \omega_i) \in \mathcal{X}_i'\}$. The random variable ω_i is independent of x and of ω_j at all other nodes, and has a finite sample space Ω_i with probability mass function (pmf) $\mathbb{P}_i(\omega)$, $\omega \in \Omega_i$. □

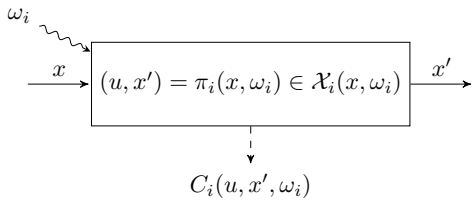

Figure 1: A schematic of a node in a policy graph. The incoming physical state is denoted by x and an incoming realization by ω_i. The decision rule $\pi_i(x, \omega_i)$ specifies the control, u, and outgoing physical state, x'. We incur a one-step cost $C_i(u, x', \omega_i)$, and transition according to one-step probabilities ϕ_{ij} (not shown) to a child node, or terminate with probability $1 - \sum_{j \in i^+} \phi_{ij}$.

Using the concept and notation of a policy graph, we can now define the optimization problem that we study in the remainder of this paper. In what follows, we assume that a common space $\mathcal{X}'$ contains $\mathcal{X}_i'$ for all $i \in \mathcal{N}$, where $\mathcal{X}'$ is compact.

Definition 2. A risk-neutral optimization problem on a policy graph is

$$\min_{\pi \in \Pi} \mathop{\mathbb{E}}_{i \in R^+;\, \omega_i} [V_{1,i}(x_R, \omega_i)], \tag{1}$$

where for $i \in \mathcal{N}$ and $t = 1, 2, \ldots$

$$V_{t,i}(x, \omega_i) = \left\{ C_i(u, x', \omega_i) + \mathop{\mathbb{E}}_{j \in i^+;\, \omega_j} [V_{t+1,j}(x', \omega_j)] \right\},$$

and where the control and outgoing physical state variables are specified by policy π's non-anticipative decision rules, which satisfy:

$$(u, x') = \pi_i(x, \omega_i) \in \mathcal{X}_i(x, \omega_i). \qquad \square$$

In Definition 2, t counts transitions from the root along a sample path, and i indexes the Markov chain's current node. In the Bellman reformulation of Section 2.2, the value function will depend only on i, with t dropping out.

A solution to problem (1) is a policy π^* that minimizes the expected cost, starting from the root. The expectation operator, $\mathbb{E}[\cdot]$, accounts for transition probabilities ϕ_{ij} from node i to nodes $j \in i^+$, and the random variable ω_j:

$$\mathop{\mathbb{E}}_{j \in i^+;\, \omega_j}[V_{t+1,j}(x', \omega_j)] = \sum_{j \in i^+} \phi_{ij} \sum_{\omega \in \Omega_j} \mathbb{P}_j(\omega) \cdot V_{t+1,j}(x', \omega).$$

When summing, or otherwise enumerating over $\omega \in \Omega_j$, we typically suppress ω's superfluous index, j, unless necessary to disambiguate.

If a policy graph is acyclic (see Example 1 below), the problem has a finite horizon. If i is a leaf node of $\mathcal{G}$ then $i^+ = \emptyset$, and so we need not specify a terminal value function. If a policy graph contains cycles (see Example 2 below), the problem has an infinite horizon. In such problems, one can minimize long-run average cost or total discounted cost. Our approach is of the latter type because the nodes $i \in \mathcal{N}$ are transient states in an absorbing Markov chain. Given an ergodic chain and a discount factor (e.g., $\rho = 0.95$) that applies at each time step, in order to use our formulation, we multiply each entry of the one-step transition matrix by ρ so that Φ_0 would become a sub-stochastic matrix for transitions among transient states of an absorbing chain.

When modeling, the information sufficient to communicate the formulation of problem (1) is: the set of nodes $\mathcal{N}$; the transition matrix Φ; the physical state at the root x_R; the random variable ω_i with pmf $\mathbb{P}_i$ for all $i \in \mathcal{N}$; and the *subproblems*, $\mathbf{SP}_i(x, \omega_i)$, defined as the constrained optimization problem:

$$\mathbf{SP}_i(x, \omega_i): \quad \min_{u, x'} \left\{ C_i(u, x', \omega_i) \;\middle|\; (u, x') \in \mathcal{X}_i(x, \omega_i) \right\}.$$

2.2 Bellman recursion reformulation

Problem (1) can be reformulated using Bellman's equation [6]:

$$\begin{aligned} V_i(x, \omega_i) = \min_{u, x'} \quad & \left\{ C_i(u, x', \omega_i) + \mathop{\mathbb{E}}_{j \in i^+;\, \omega_j}[V_j(x', \omega_j)] \right\} \\ \text{s.t.} \quad & (u, x') \in \mathcal{X}_i(x, \omega_i), \end{aligned} \tag{2}$$

and an optimal policy π^* can be obtained by choosing a control u in the argmin of problem (2):

$$\pi_i(x, \omega_i) \in \mathop{\arg\min}_{(u, x') \in \mathcal{X}_i(x, \omega_i)} \left\{ C_i(u, x', \omega_i) + \mathop{\mathbb{E}}_{j \in i^+;\, \omega_j}[V_j(x', \omega_j)] \right\}.$$

For an acyclic policy graph, i.e., a finite-horizon problem, the mapping between formulations (1) and (2) is straightforward. Unrolling recursion (2) on an acyclic graph allows i in (2) to encode i and t in (1). (See discussion in subsequent Example 2.) For a cyclic policy graph, i.e., an infinite-horizon problem, the reformulation is justified by considering the Bellman operator $\mathcal{B}$:

$$\begin{aligned} \mathcal{B}v_i(x, \omega_i) = \min_{u, x'} \quad & \left\{ C_i(u, x', \omega_i) + \mathop{\mathbb{E}}_{j \in i^+;\, \omega_j}[v_j(x', \omega_j)] \right\} \\ \text{s.t.} \quad & (u, x') \in \mathcal{X}_i(x, \omega_i), \end{aligned} \tag{3}$$

which applies to the space of bounded functions, v, on $\mathcal{D} \equiv \cup_{i \in \mathcal{N}}(\mathcal{X}' \times \{i\} \times \Omega_i)$. Let $\|v\|_\infty = \sup_{(x,i,\omega) \in \mathcal{D}} |v_i(x, \omega)|$, and let $\gamma \in (0, 1)$. We say that $\mathcal{B}$ is a γ-contraction if for any pair of bounded functions v and v' on domain $\mathcal{D}$ we have $\|\mathcal{B}v - \mathcal{B}v'\|_\infty \leq \gamma \|v - v'\|_\infty$.

Writing the fixed-point condition $\mathcal{B}V = V$ is synonymous with equation (2) holding for all $(x, i, \omega_i) \in \mathcal{D}$. If $\mathcal{B}$ a γ-contraction, then $\mathcal{B}V = V$ has a unique fixed point, and solving the infinite horizon problem (1) is equivalent to solving the fixed-point equation (2). In this case, the Bellman policy is optimal, so that:

$$\min_{\pi \in \Pi} \mathop{\mathbb{E}}_{i \in R^+;\, \omega_i}[V_{t,i}(x_R, \omega_i)] = \mathop{\mathbb{E}}_{i \in R^+;\, \omega_i}[V_i(x_R, \omega_i)].$$

Although our Bellman reformulation of a cyclic policy graph problem (1) hinges on a value-function argument, the policy graph is a *modeling framework* that is independent of the solution method. In this paper, we compute policies using SDDP-style algorithms, i.e., value-function approximation algorithms, but given a policy graph, one could apply other algorithms.

2.3 Assumptions

To ensure our formulation is well-defined and to facilitate computationally tractable algorithms, we make the following assumptions that restrict the space of policy graphs that we consider:

(A1) The number of nodes, $\mathcal{N}$, is finite.

(A2) The sample space, Ω_i, is finite at each node $i \in \mathcal{N}$, and the random variables ω_i, $i \in \mathcal{N}$, are mutually independent.

(A3) For $i \in \mathcal{N}$, $\mathcal{X}_i(x, \omega_i) = \{(u, x') : u \in U_i(x, \omega_i),\ x' = T_i(u, x, \omega_i) \in \mathcal{X}_i'\} \neq \emptyset$ and $U_i(x, \omega_i)$ is compact $\forall x \in \mathcal{X}'$, $\omega_i \in \Omega_i$, where $\mathcal{X}_i' \subseteq \mathcal{X}'$ and where $\mathcal{X}'$ is compact.

(A4) $\min_{u,x',x} \left\{ C_i(u, x', \omega_i) \;\middle|\; (u, x, x') \in \mathcal{X}_i(\omega_i) \equiv \{(u, x, x') : (u, x') \in \mathcal{X}_i(x, \omega_i)\} \right\}$ is a convex optimization model for each $i \in \mathcal{N}$ and $\omega_i \in \Omega_i$.

(A5) The one-step matrix Φ_0 is such that either the policy graph is acyclic or the Bellman operator $\mathcal{B}$ in equation (3) is a γ-contraction.

In addition to (A1) and (A2), the algorithms we develop in this paper require that $|\Omega_i|$ and $|\mathcal{N}|$ are of modest size, e.g., at most 100 (to an order of magnitude). Under (A3)—which includes relatively complete recourse—subproblem $\mathbf{SP}_i(x, \omega_i)$ is feasible, and the subproblem further has a finite optimal solution because its objective function is convex by (A4). Assumption (A4) further ensures that problem (1) is convex with respect to x and u. As we discuss in Section 2.2, (A5) ensures that the Bellman reformulation is equivalent to Section 2.1's model. That said, in the infinite-horizon setting, we want to be able to verify that (A5) holds in terms of our problem's primitives. Given our assumptions, including compactness of $\mathcal{U}_i(x, \omega_i)$ and $\mathcal{X}'$, a sufficient condition on the one-step transition probabilities, Φ_0, to ensure (A5) holds is that $\gamma \equiv \max_{i \in \mathcal{N}} \sum_{j \in i^+} \phi_{ij} < 1$. In this case, a standard argument shows that $\mathcal{B}$ is a γ-contraction.

2.4 Policy graphs and MDPs

Section 1 denotes the state by $s \in \mathcal{S}$, which is the pervasive convention in discrete-action discrete-state MDPs. The analogous state in a policy graph for our CACS MDP is (x, i), where x is a vector of continuous physical state variables (we allow some discrete variables in what follows) and i is one of a finite number of Markov-chain states. As indicated, $(u, x') \in \mathcal{X}_i(x, \omega_i)$ is shorthand for $u \in U_i(x, \omega_i)$ and $x' = T_i(u, x, \omega_i) \in \mathcal{X}_i'$. Thus $U_i(x, \omega_i)$ and $\mathcal{X}_i'$ constrain the control—the analog of $a \in \mathcal{A}_s$ in our typical MDP notation—and given u, $T_i(u, x, \omega_i)$ determines the outgoing state. In this way, $x' = T_i(u, x, \omega_i)$ specifies a (conditionally) deterministic transition for the x-component of the state, and our one-step transition matrix Φ specifies a probabilistic transition for the i-component of the state. Section 1 points to a distribution, $\mu(s)$, on initial states. Consistent with the transitions in a policy graph just mentioned, we initialize the root node with a deterministic physical state, x_R, but allow a random transition from the root, $\phi_{R,i}$, which is the analog of $\mu(s)$ but only for the i-component of the state.

Computation with MDPs is typically accomplished using discrete action and state spaces. A policy graph, combined with SDDP, offers an alternative. Exploiting the convexity assumptions of Section 2.3, we can handle continuous states and actions directly, constructing a piecewise linear outer approximation of the cost-to-go function without discretization.

2.5 Policy graphs and MSPs

A multi-stage stochastic program with a finite horizon, finite set of scenarios, and general inter-stage dependence is typically defined on a scenario tree. Such a model is a special sub-class of policy graphs in which the matrix Φ defines a tree and the random variable at each node is degenerate, i.e., $|\Omega_i| = 1$. Here, the nodes and arcs of the scenario tree correspond exactly to the nodes and arcs of the equivalent policy graph. If the MSP is inter-stage independent it could again be represented as just sketched, but with Φ and the degenerate realizations at each node encoding inter-stage independence. That said, policy graphs afford an attractive and compact alternative, illustrated in Figure 2.

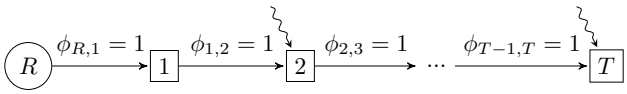

Figure 2: Policy graph structure for a standard T-stage MSP with inter-stage independence. We refer to this as a *linear* policy graph.

In this way, all problems defined on a finite scenario tree can be captured using policy graphs, but the converse is false. Figure 3 depicts what would be a four-stage stochastic program, except that return arcs with probability $\rho < 1$ yield an infinite-horizon problem. This example is defined using an (infinite-horizon) Markov process rather than a finite scenario tree. The next subsection begins to develop such examples in more detail.

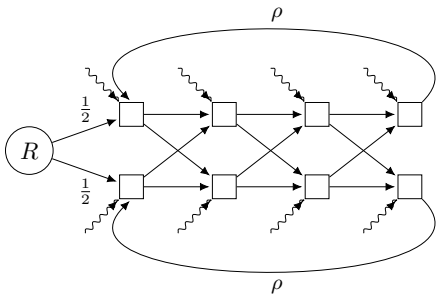

Figure 3: In each fiscal quarter, we can be in a volatile or normal state (top and bottom rows). Each is equally likely initially, and there is Markov switching each quarter, meaning the agent faces a mixture model. Further transition probabilities are suppressed except those that return from the final quarter of one year to the first quarter of the next, with probability $\rho < 1$. (For simplicity we do not show switching for this return.) This is a cyclic policy graph.

A policy graph with a node i for which $|\Omega_i| > 1$ can be expanded into a larger policy graph with an extra node for each realization of ω_i. Conversely, two nodes in a policy graph can be combined if they share identical expected cost-to-go functions and identical functional forms of the constraint set; see (2). A computational advantage of a parsimonious policy graph, i.e., one with fewer nodes, is that SDDP-style algorithms can then exploit shared expected cost-to-go functions.

2.6 Examples

We provide examples that begin to demonstrate the modeling flexibility of policy graphs. For clarity, we present minimal instances that isolate essential elements of our approach. These examples are not merely illustrative. Rather, they are canonical building blocks that recur as sub-structures in large-scale models, and the modeling insights developed here extend to those settings.

Example 1 (Two-stage newsvendor). Consider a two-stage newsvendor problem. In the first stage, the agent decides the number of newspapers to order, which cost \$2 each. In the second stage, the agent observes demand ω and can sell newspapers at a unit price of \$5. Excess newspapers must be disposed of at a unit cost of \$0.1. Figure 4 shows the graph structure and subproblems. The problem can also be formulated as shown in Figure 5, demonstrating how scenario trees from stochastic programming are a special case of a policy graph.

Example 2 (Cyclic newsvendor). We can extend Example 1 to an infinite-horizon by adding a cyclic second-stage subproblem, which includes both buying u_b and selling u_s. The disposal cost of \$0.1 is now a holding cost. Figure 6 shows the graph structure and subproblems. We can form a finite-horizon approximation by "unrolling" the loop to form a T-stage problem as in Figure 2, except that $\phi_{2,3} = \cdots = \phi_{T-1,T} = \rho$.

Example 3 (Markovian newsvendor). Suppose that there are two market states, sunny and cloudy, and transitions between them over time can be modeled as a Markov chain. When the weather is sunny, demand

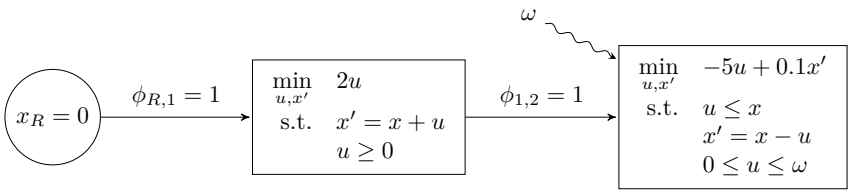

Figure 4: The policy graph for Example 1. While the problem is formulated with continuous actions and physical state, in some problem variants we instead require integer-valued decisions, e.g., $u \in \mathbb{Z}_+$ at node 1 and $u \in \{0, 1, \ldots, \omega\}$ at node 2.

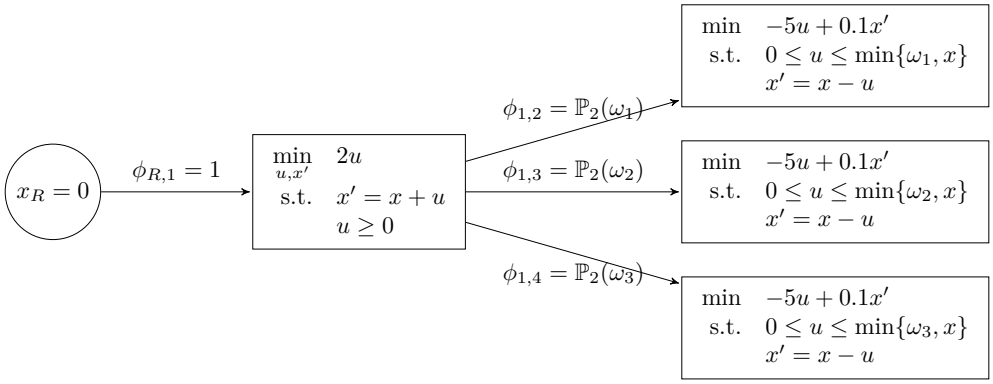

Figure 5: The policy graph for Example 1, expanded into a scenario tree with $\Omega_2 = \{\omega_1, \omega_2, \omega_3\}$. Note that the ω_i in the second-stage problems are constants, i.e., degenerate random variables, and their corresponding probability masses are now on the arcs.

is governed by a favorable random variable ω_s, and when it's cloudy, the random demand is instead ω_c. The subproblems are the same as in node 2 of Example 2, and the root state is again $x_R = 0$. The graph structure is shown in Figure 7.

3 Decision-dependent transitions

3.1 Mathematical model

In an MDP, the one-step transition probabilities, $\mathbb{P}(s'|s, a)$, depend on the agent's action. In our policy graph formulation, the next state (x', j), depends on $x' = T_i(u, x, \omega_i)$ and ϕ_{ij}. The former deterministic transition depends on the agent's action, but the latter probabilistic transition does not. Restated, our formulation of Section 2.1 has one-step transition probabilities, Φ, which do not depend on the agent's decisions.

In this section we extend policy graphs to allow action-dependent matrices, $\Phi(y)$, where y is an additional decision variable. In the stochastic programming literature, this is one form of decision-dependent uncertainty [2, 3, 16, 25].

We give two formulations to handle action-dependent one-step transition matrices. Each formulation lends itself to a different SDDP variant, which we discuss in Section 6. In our first decision-dependent model we have:

$$\min_{\pi \in \Pi} \sum_{i \in R^+} \phi_{R,i} \sum_{\omega \in \Omega_i} \mathbb{P}_i(\omega) \cdot V_i(x_R, \omega),$$

where:

$$\begin{aligned} V_i(x, \omega_i) = \min_{u, x', y} \quad & C_i(u, x', \omega_i) + C_i'(y) + \sum_{j \in i^+; \omega \in \Omega_j} \phi_{ij}(y) \cdot \mathbb{P}_j(\omega) \cdot V_j(x', \omega) \\ \text{s.t.} \quad & (u, x') \in \mathcal{X}_i(x, \omega_i) \\ & y \in \mathcal{Y}. \end{aligned} \tag{4}$$

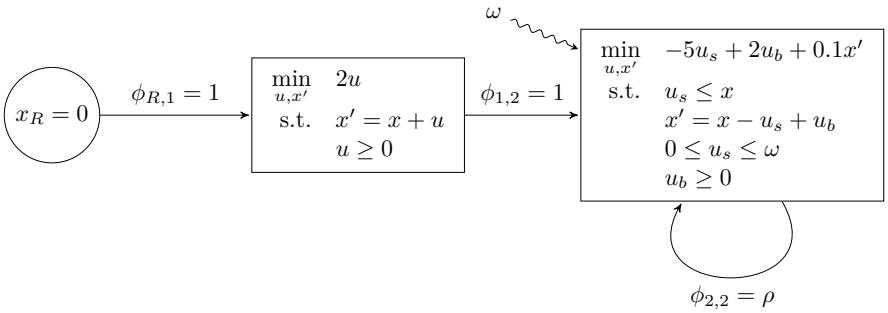

Figure 6: The policy graph for Example 2. The same comments from Example 1 hold regarding continuity of u.

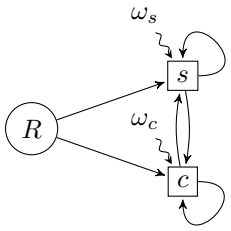

Figure 7: The policy graph for Example 3. When the process exits the sunny node, it returns with probability $\phi_{s,s}$, transitions to the cloudy node with probability $\phi_{s,c}$, and the process terminates with probability $1 - \phi_{s,s} - \phi_{s,c}$. Analogous probabilistic transitions occur out of the cloudy market state.

Decision y could represent marketing, which increases the likelihood of a favorable demand distribution. Such a decision incurs a cost $C_i'(y)$, which we assume is linear and independent of the control decisions. We assume $0 \in \mathcal{Y}$, where $\mathcal{Y}$ is a polytope, and for simplicity, we assume identical constraints $y \in \mathcal{Y}$ at each node i. In a more general model, y could compete with u for limited resources and the set $\mathcal{Y}$ may differ between nodes. For the one-step transition matrices, we assume a linear response to y:

$$\phi_{ij}(y) = \phi_{ij}^0(1 + a_j^\top y), \text{ for } i, j \in \mathcal{N}, \tag{5}$$

where ϕ_{ij}^0 is a nominal one-step transition probability when $y = 0$. We then require $(1 + a_j^\top y) \geq 0,\ \forall y \in \mathcal{Y}$, and $\sum_{j \in i^+} \phi_{ij}^0 a_j = 0$. Because $C_i'(y)$ and $\phi_{ij}(y)$ are both linear in y, we can restrict attention to the extreme points of $\mathcal{Y}$ in the minimization of (4). In what follows we will assume $\mathcal{Y}$ is finite with a modest number of options at each node in the policy graph. Even in this setting, there are a huge number of such options in the overall problem.

Our second formulation is motivated by the above development but allows greater generality. We assume a limited set of possible transition matrices $\{\Phi^d\}_{d \in D}$, and at each node, a binary variable $y_d \in \{0, 1\}$ specifies the matrix for the next step. The cost-to-go function at node $i \in \mathcal{N}$ is now:

$$\begin{aligned} V_i(x, \omega_i) = \min_{u,x',y} \quad & C_i(u, x', y, \omega_i) + \sum_{d \in D} y_d \sum_{j \in i^+;\, \omega \in \Omega_j} \phi_{ij}^d \cdot \mathbb{P}_j(\omega) \cdot V_j(x', \omega) \\ \text{s.t.} \quad & (u, x', y) \in \mathcal{X}_i(x, \omega_i) \\ & \sum_{d \in D} y_d = 1 \\ & y_d \in \{0, 1\}, \quad d \in D. \end{aligned} \tag{6}$$

In an equivalent reformulation of (4) we would have constraints $(u, x') \in \mathcal{X}_i(x, \omega_i)$, but we use the more general $(u, x', y) \in \mathcal{X}_i(x, \omega_i)$ to allow coupling between u and y. We also allow for a more general form $C_i(u, x', y, \omega_i)$ for reasons that will become clear in Section 6.2.2.

Problem formulations with recursions (4) and (6) bring computational challenges because the products between y (or y_d) and $V_j(x', \omega_j)$ induce non-convexities. In particular, assume the cost-to-go function is

convex for fixed y. Then, optimizing over y yields the minimum of a set of convex functions, i.e., it yields a non-convex, but piecewise convex cost-to-go function. We revisit this issue via two convex relaxations in Section 6.2, but first we turn to examples.

3.2 Examples

To demonstrate the flexibility of our decision-dependent modeling approach, we present a second set of examples.

Example 4 (The cheese producer). Consider a small farm that produces cheese. Due to the weather and biological variation, the quantity of cheese produced by the farm each week is the random variable ω_q [kg]. Each weekend, the farmer can sell cheese at Sunday's market, which has stochastic demand. Any unsold cheese is stored in inventory for the following week. Attending the market costs c_y [\$], and the cheese sells for c_u [\$/kg]. If the farmer attends the market, the demand for cheese is the random variable ω_d [kg]. The policy graph formulation is given in Figure 8.

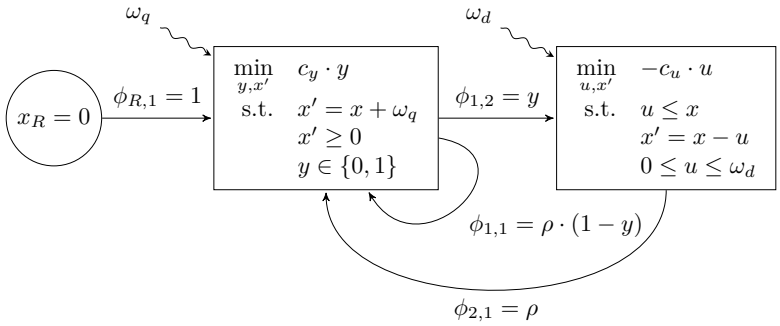

Figure 8: The policy graph for Example 4. This example is a variant of the cyclic newsvendor problem in Examples 2 and 3, with random production amounts, along with a binary decision that dictates at which inventory levels we transition to market, i.e., transition to node 2.

We can form equivalent models for Example 4 that do not use decision-dependent transition matrices. For example, we could have added y as a state variable with the constraint $0 \leq u \leq \omega_d \cdot y$. In general, it is always possible to express a decision-dependent model as a decision-independent model with an expanded state space. We advocate using the decision-dependent policy graph because of the convenience and clarity that it provides the modeler.

Example 5 (The advertising cheese producer). Consider the following variation of Example 4, in which the farmer always attends Sunday's market. During the week, the farmer can advertise for a fixed cost of c_y [\$]. If the farmer advertises, the random demand for cheese is ω_d^H, and otherwise the demand is ω_d^L. The policy graph formulation is given in Figure 9.

Example 6 (Advertising). Consider a Markovian newsvendor problem from Example 3, with the nodes corresponding to *high* and *low* distributions of demand rather than sunny and cloudy conditions, respectively. In each week, the agent can market the product via an advertising budget $y \in \{0, 1\}$. The evolution of the market state forms a Markov chain with a one-step transition matrix that depends on y and includes the discount factor $0 < \rho < 1$:

$$\Phi(y) = \rho \cdot \begin{bmatrix} (0.5 + 0.3y) & (0.5 - 0.3y) \\ (0.5 + 0.2y) & (0.5 - 0.2y) \end{bmatrix}.$$

4 Learning

4.1 Mathematical model

When we must simultaneously learn the reward and transition functions as well as the policy, the requisite statistical learning can occur in a model-based or model-free framework. Here, we consider a setting that

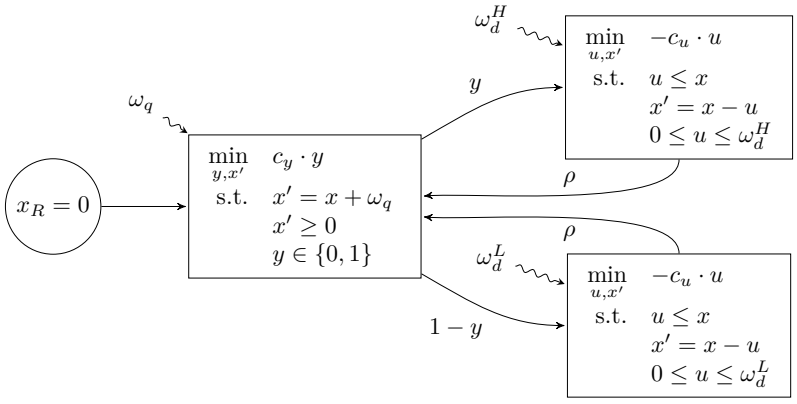

Figure 9: The policy graph for Example 5. This variant of Example 4 uses a binary variable y to determine inventory levels at which we should advertise to produce a favorable distribution of demand, ω_d^H versus ω_d^L, at the market.

requires learning the probability distribution that governs the costs we incur (or, rewards we receive), and the one-step transition matrix. We do so in a structured way, even relative to typical model-based contexts. Once the policy graph model is formulated, our development adapts ideas from Dowson et al. [15] for partially observable multi-stage stochastic programs. This section's model also sets up the extension of Section 5.1 in which we combine decision-dependent transitions and statistical learning.

We posit a set of candidate models indexed by $m \in \mathcal{M}$, and we assign a prior belief $b_m > 0$, $m \in \mathcal{M}$, that each is correct, $\sum_{m \in \mathcal{M}} b_m = 1$. For computational tractability (see Section 6.3's algorithm), $|\mathcal{M}|$ should be small. Each model is defined on an identical set of policy graph nodes, denoted by $\mathcal{N}$. Model $m \in \mathcal{M}$ has an $|\mathcal{N}| \times |\mathcal{N}|$ one-step transition matrix Φ_0^m, along with $\phi_{R,i}^m$, $i \in \mathcal{N}$, specifying the initial transition probabilities from the root. Model m has a pmf governing the randomness at each node, $i \in \mathcal{N}$, denoted by $\mathbb{P}_i^m(\omega)$, $\omega \in \Omega_i$. We assume that while the one-step transition probabilities can differ, the locations of the nonzero entries are identical across Φ_0^m, $m \in \mathcal{M}$. Moreover, for each model $m \in \mathcal{M}$, while the pmfs can differ, their possible realizations are identical, i.e., they do not depend on m, as the notation Ω_i indicates.

To enable statistical learning, we form a larger, augmented policy graph, denoted $\mathcal{G} = (R, \mathcal{N} \times \mathcal{M}, \Phi, \mathcal{A})$, which we now describe. We clone nodes across the set of models via $(i, m) \in \mathcal{N} \times \mathcal{M}$. We use a single root node, R. The first row of the $|\mathcal{N} \times \mathcal{M}| + 1$ by $|\mathcal{N} \times \mathcal{M}|$ matrix Φ is defined by the product $b_m \phi_{R,i}^m$, $m \in \mathcal{M}$ and $i \in \mathcal{N}$, which captures the prior likelihood of each model and the initial transition probabilities within each model. The remaining rows of Φ are defined by the block-diagonal matrix, $\text{diag}\,(\Phi_0^m : m \in \mathcal{M})$. The final construct needed is to partition the nodes $\mathcal{N} \times \mathcal{M}$ into *ambiguous subsets*, $A_i = \{(i, m) : m \in \mathcal{M}\}$ so that $\bigcup_{i \in \mathcal{N}} A_i = \mathcal{N} \times \mathcal{M}$. The agent can observe the current ambiguity set, A_i, which is equivalent to simply observing i of the pair (i, m). However, the agent cannot distinguish the models, $m \in \mathcal{M}$, within that ambiguity set. The policy graph $\mathcal{G}$ is augmented by the ambiguity sets, where $\mathcal{A}$ is the partition of the nodes $\mathcal{N} \times \mathcal{M}$ defined by A_i, $i \in \mathcal{N}$.

The dynamics in a *fully observable* (i.e., a typical) policy graph are as follows. The agent enters node i knowing: (a) the previous node, j; (b) the incoming physical state, x; (c) the observation, $\omega_i \in \Omega_i$; (d) the structural constraints, $(u, x') \in \mathcal{X}_i(x, \omega_i)$; (e) the one-step cost, $C_i(u, x', \omega_i)$; (f) the one-step transition probabilities departing i, ϕ_{ij} for $j \in i^+$ (and subsequent nodes); and (g) the pmf $\mathbb{P}_j(\omega)$, $\omega \in \Omega_j$ for $j \in i^+$ (and subsequent nodes).

In our current setting the agent instead enters ambiguity set A_i knowing: (a) the previous ambiguity set, A_j; (b) the incoming physical state, x; (c) the observation, $\omega_i \in \Omega_i$; (d) the structural constraints, $(u, x') \in \mathcal{X}_i(x, \omega_i)$; and (e) the one-step cost, $C_i(u, x', \omega_i)$. As the notation indicates, Ω_i, $\mathcal{X}_i(x, \omega_i)$, and $C_i(u, x', \omega_i)$ are each common across all nodes in the ambiguity set $(i, m) \in A_i$. In contrast to the fully observable setting, the agent knows neither (f) nor (g), but has a belief pmf as to which model is correct. Thus the observation $\omega_i \in \Omega_i$ in (c) now comes from a mixture of $\mathbb{P}_i^m$ with belief weights b_m, $m \in \mathcal{M}$. That belief is continually updated via observations. In particular, at the prior stage when in ambiguity set A_j, let the agent's belief pmf be b_m, $m \in \mathcal{M}$. Upon observing the transition from A_j to A_i and observing $\omega_i \in \Omega_i$

the agent updates the belief from b to b' using Bayes' theorem:

$$\begin{aligned} b'_m &= \mathbb{P}(m \,|\, \omega_i \text{ and } j \to i) \\ &= \frac{\mathbb{P}(\omega_i \text{ and } j \to i \,|\, m) \cdot \mathbb{P}(m)}{\mathbb{P}(\omega_i \text{ and } j \to i)} \\ &= \frac{\mathbb{P}_i^m(\omega_i) \cdot \phi_{ji}^m \cdot b_m}{\sum_{m \in \mathcal{M}} \mathbb{P}_i^m(\omega_i) \cdot \phi_{ji}^m \cdot b_m}. \end{aligned} \tag{7}$$

We use "m" to denote the event that $m \in \mathcal{M}$ is the correct model and "$j \to i$" to denote the event that we transitioned from A_j to A_i. We let $b' = B(b, j \to i, \omega_i)$ denote the vector-valued update obtained from equation (7).

Model (1)'s Bellman reformulation can now be expressed as follows:

$$\min_{\pi \in \Pi} \sum_{m \in \mathcal{M}} b_m \sum_{i \in R^+} \phi_{R,i}^m \sum_{\omega \in \Omega_i} \mathbb{P}_i^m(\omega) \cdot V_i(x_R, B(b, R \to i, \omega), \omega), \tag{8}$$

where:

$$\begin{aligned} V_i(x, b, \omega_i) = \min_{u, x'} \quad & C_i(u, x', \omega_i) + \mathcal{V}_i(x', b) \\ \text{s.t} \quad & (u, x') \in \mathcal{X}_i(x, \omega_i), \end{aligned} \tag{9}$$

and where:

$$\mathcal{V}_i(x', b) = \sum_{m \in \mathcal{M}} b_m \sum_{j \in i^+} \phi_{ij}^m \sum_{\omega \in \Omega_j} \mathbb{P}_j^m(\omega) \cdot V_j(x', B(b, i \to j, \omega), \omega).$$

Thus, we have formulated a policy graph model with ambiguous subsets that captures competing models, $m \in \mathcal{M}$, of our primitives, i.e., competing models of the one-step transition matrix and the randomness at each policy graph node. This formulation incorporates statistical learning into the policy graph framework.

The simplicity of the Bayesian update (7) relative to that in [15] comes from the model structure. If we had a Markov switching model—in which transitions among models $m \in \mathcal{M}$ were possible over time—then we would not have the block-diagonal structure within Φ discussed above, and the numerator in (7) would involve a sum across multiple models.

Under model misspecification, the update from Bayes' theorem (7) can become overconfident in the wrong model. A strategy to avoid that is tempering, which forms a convex combination of the Bayesian update and the previous belief b_m [10, 27, 40]. The convex combination's weight specifies a learning rate, which can be made adaptive to account for misspecification or nonstationarity [22]. Our framework could accommodate such an update in case robustness considerations are needed.

There is analogous work in the MDP literature involving unknown transition probabilities, hidden context, and competing models [8, 19, 24, 50], and via control of time-staged hidden Markov models, e.g., [17], that capture, for example, hidden bear-neutral-bull states of the market in financial applications [38]. These ideas inspire what we propose here for our class of MDPs, which we can solve using adaptations of SDDP; see Section 6.

4.2 Examples

An example motivates our approach to statistical learning. We defer further examples to Section 5 where we formulate models with decision-dependent learning.

Example 7 (Markovian newsvendor with learning). In Example 3, we sketch a newsvendor model in which we have Markov switching between sunny and cloudy states, with a one-step transition matrix,

$$\Phi = \begin{bmatrix} \phi_{R,s} & \phi_{R,c} \\ \phi_{s,s} & \phi_{s,c} \\ \phi_{c,s} & \phi_{c,c} \end{bmatrix}.$$

We assume all entries of Φ are positive, $\phi_{s,s} + \phi_{s,c} < 1$, and $\phi_{c,s} + \phi_{c,c} < 1$, so that assumption (A5) holds. Here, we extend the example to the statistical learning setting of Section 4.1 using Figure 10. The left and

right submodels clone the underlying policy graph using two candidate models, $\mathcal{M} = \{1, 2\}$, with prior belief $b_1 > 0$ and $b_2 > 0$, $b_1 + b_2 = 1$. With new node set $\mathcal{N} \times \mathcal{M} = \{(s,1), (c,1), (s,2), (c,2)\}$, the one-step transition matrix is:

$$\Phi = \begin{bmatrix} b_1\phi^1_{R,s} & b_1\phi^1_{R,c} & b_2\phi^2_{R,s} & b_2\phi^2_{R,c} \\ \phi^1_{s,s} & \phi^1_{s,c} & 0 & 0 \\ \phi^1_{c,s} & \phi^1_{c,c} & 0 & 0 \\ 0 & 0 & \phi^2_{s,s} & \phi^2_{s,c} \\ 0 & 0 & \phi^2_{c,s} & \phi^2_{c,c} \end{bmatrix},$$

where the superscripts 1 and 2 correspond to the two models. The ambiguity sets are $A_s = \{(s,1), (s,2)\}$ and $A_c = \{(c,1), (c,2)\}$, so the agent is aware whether it is sunny or cloudy but is unaware whether the correct model is left (1) or right (2). The agent sequentially observes the transitions among the sunny and cloudy nodes and observes realizations of demands at those nodes, which have common support across the two models but have different pmfs. At each time step the agent updates the belief pmf, b, via equation (7), learning which model is correct.

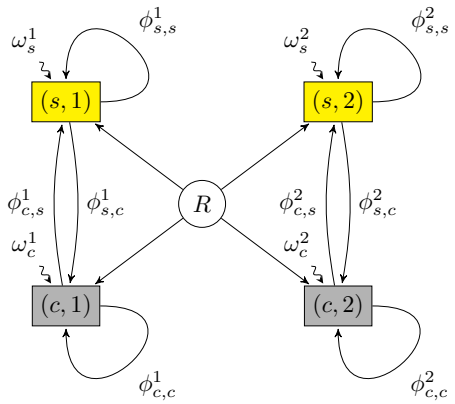

Figure 10: The policy graph for Example 7. The agent is aware of whether the current node is sunny (yellow) or cloudy (gray), but is unaware of whether model $m = 1$ (left) or $m = 2$ (right) is correct. The one-step transition probabilities differ in the left and right halves of the graph, as illustrated by the Φ^1 and Φ^2 labels on arcs. While the supports in the two models are identical, their pmfs differ. The agent observes transitions between sunny and cloudy ambiguity sets, observes demands, and iteratively updates the belief pmf using Bayes' rule (7).

5 Decision-dependent learning

5.1 Mathematical model

We formulate a decision-dependent learning model, combining ideas from Sections 3.1 and 4.1. That is, we allow for action-dependent one-step transition probabilities, which can inform our statistical learning of which probabilistic model is correct:

$$\min_{\pi \in \Pi} \sum_{m \in \mathcal{M}} b_m \sum_{i \in R^+} \phi^m_{R,i} \sum_{\omega \in \Omega_i} \mathbb{P}^m_i(\omega) \cdot V_i(x_R, B_R(b, R \to i, \omega), \omega) \tag{10}$$

where:

$$\begin{aligned} V_i(x, b, \omega_i) = \min_{u, x', y} \quad & C_i(u, x', y, \omega_i) + \mathcal{V}_i(x', y, b) \\ \text{s.t} \quad & (u, x') \in \mathcal{X}_i(x, \omega_i) \\ & y \in \mathcal{Y}, \end{aligned} \tag{11}$$

and where:

$$\mathcal{V}_i(x', y, b) = \sum_{m \in \mathcal{M}} b_m \sum_{j \in i^+} \phi_{ij}^m(y) \sum_{\omega \in \Omega_j} \mathbb{P}_j^m(\omega) \cdot V_j(x', B(y, b, i \to j, \omega), \omega).$$

Here, $B_R(b, R \to i, \omega)$ has the form of equation (7) because there is no y-decision at the root node, and $B(y, b, j \to i, \omega_i)$ is the vector form of:

$$b'_m \quad = \quad \frac{\mathbb{P}_i^m(\omega_i) \cdot \phi_{ji}^m(y) \cdot b_m}{\sum_{m \in \mathcal{M}} \mathbb{P}_i^m(\omega_i) \cdot \phi_{ji}^m(y) \cdot b_m}. \tag{12}$$

We update the belief pmf given that we have observed a transition from ambiguity set A_j to A_i, observed realization ω_i in A_i, and given that decision y was made in node j, affecting the transition probabilities via $\phi_{ji}^m(y)$.

5.2 Examples

We present three examples, which help motivate the formulation above.

Example 8 (Marketing discovery). Let ϕ_{ij} capture transitions between favorable and less favorable states in a Markov-switching model. Thus, as discussed in Example 6, we can think of y as a marketing decision that may be used to increase the likelihood, via $\phi_{ij}(y)$, of transitioning to a favorable state. Now, however, we may be unsure of the effectiveness of marketing, with it being more effective under model $m = 1$, $\phi_{ij}^1(y)$, and less effective under $m = 2$, $\phi_{ij}^2(y)$. Moreover, we may be unsure of *how* favorable the demand distribution $\mathbb{P}^1$ is relative to $\mathbb{P}^2$. By investing in potentially costly marketing, we can learn the effectiveness and degree of favorability. An example decision-dependent one-step transition matrix is:

$$\Phi(y) = \rho \cdot \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ (0.5 + 0.3y) & (0.5 - 0.3y) & 0 & 0 \\ (0.5 + 0.2y) & (0.5 - 0.2y) & 0 & 0 \\ 0 & 0 & (0.5 + 0.1y) & (0.5 - 0.1y) \\ 0 & 0 & (0.5 + 0.1y) & (0.5 - 0.1y) \end{bmatrix},$$

where we have a binary marketing decision $y \in \mathcal{Y} = \{0, 1\}$ at each node.

Example 9 (Decision-dependent information discovery). Consider the following variation of Example 4. When starting their business, the farmer has two hypotheses for demand for cheese, represented by the distributions of ω_d^H and ω_d^L, which are equally likely. Each week, the farmer can pay to attend the market and observe one demand realization. With this information, the farmer updates their belief pmf for whether the demand is from ω_d^H or ω_d^L. If the farmer does not attend the market, no new information is learned, and the belief pmf is not updated. The policy graph formulation is given in Figure 11.

Example 10 (Tiger problem). Consider the following problem from Section 5.1 of [28]. An agent is in a room with two exit doors. Behind one is a tiger and the other is escape. In each period, the agent may listen for the tiger or open one of the doors. Opening the door to the tiger costs \$100 (in medical bills). Finding the escape is a reward of \$10. Each turn listening costs \$1. Because of the muffled sound, if the agent listens for the tiger and it appears to come from a particular door, there is a 15% chance that it is a false positive and the tiger is behind the other door. The true-positive and false-positive events are independent across periods. Thus, at the per-period listening cost, the agent can improve estimation of the tiger's location by listening in multiple periods.

The problem can be cast as our decision-dependent learning model (10); see Figure 12. There are no state or control variables in the model, only the decision-dependent y. The objective function of the l and r nodes is the constant 1 (representing the cost of listening), the objective function of the ll and rr nodes is the constant 100 (representing opening the door to the tiger), and the objective function of the lr and rl nodes is the constant -10 (representing opening the door to escape).

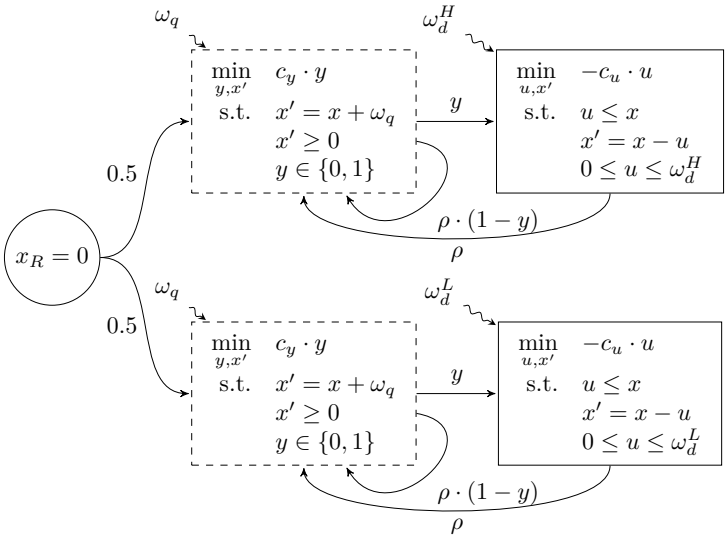

Figure 11: The policy graph for Example 9. The two dashed nodes form an ambiguity set, and the two solid nodes form an ambiguity set.

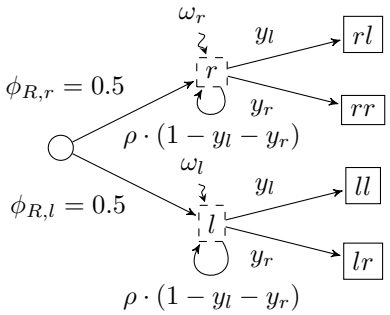

Figure 12: The tiger problem formulated as a decision-dependent learning model (10) with $y_l + y_r \leq 1, y_l, y_r \in \{0,1\}$. The ambiguity set is $\mathcal{A} = \{\{l,r\},\{rl\},\{rr\},\{ll\},\{lr\}\}$.

6 Stochastic dual dynamic programming

This section outlines stochastic dual dynamic programming (SDDP), the algorithm that we use to approximately solve the models we propose. We begin with the basic algorithm, which extends nested Benders' decomposition to problems with many stages and has been extensively studied (see, for example, the recent review of [18]), and which can handle the formulation of Section 2.1. We then provide new extensions of that algorithm to cover models with decision-dependent transitions and decision-dependent learning. We study SDDP algorithms because under convexity assumptions they asymptotically find optimal policies to problems with continuous action and state variables without needing discretization [42]. We say *approximately solve* for two reasons. Even under convexity, we must terminate the sampling-based algorithms finitely, but more importantly, the extensions we propose introduce non-convexities that violate assumptions required for global convergence.

6.1 SDDP basics

By assumptions (A1)–(A5) from Section 2.3, the expected cost-to-go function in problem (2) is convex in x' and hence can be approximated from below by the maximum of a set of affine functions known as *cuts*:

$$\mathop{\mathbb{E}}_{j \in i^+;\, \omega_j} \left[V_j(x', \omega_j) \right] \geq \max_{k \in \mathcal{K}} \left\{ \alpha_{i,k} + \beta_{i,k}^\top x' \right\},$$

where $\mathcal{K} = \{0, 1, \ldots, K\}$. We assume $\alpha_{i,0} = -M$ and $\beta_{i,0} = 0$ to preclude unboundedness of the subproblems we build, where M is sufficiently large.

SDDP iteratively forms cuts that give a polyhedral outer-approximation of the expected cost-to-go function at each node $i \in \mathcal{N}$, exploiting independence of ω_i, $i \in \mathcal{N}$, and the Markovian nature of the one-step transitions. After K iterations of SDDP we approximate (2) as follows:

$$\begin{aligned} \mathbf{SP}_i^K: \quad V_i^K(\bar{x}, \omega_i) = \min_{u,x,x',\theta} \quad & C_i(u, x', \omega_i) + \theta \\ \text{s.t.} \quad & (u, x, x') \in \mathcal{X}_i(\omega_i) \\ & \theta \geq \alpha_{i,k} + \beta_{i,k}^\top x', \qquad k \in \mathcal{K} \\ & x = \bar{x} \qquad\qquad [\lambda]. \end{aligned}$$

The constraints include $(u, x, x') \in \mathcal{X}_i(\omega)$ (see assumption (A4)) because we added constraint $x = \bar{x}$. This *fishing constraint* both simplifies assessing convexity and computing cuts because the incoming state, $\bar{x}$, appears only in the right-hand side of a linear constraint. Thus, an optimal dual variable λ is a subgradient of $V_i^K(\cdot, \omega_i)$ at $\bar{x}$.

Each iteration of SDDP has two phases: (i) a forward pass, which, by sampling a sequence of nodes and realizations of the random variables, generates a sequence of state variables; and (ii) a backward pass, which constructs a new cut at each value of the state variables from the forward pass.

Algorithm 1 gives pseudo-code for SDDP's K^{th} iteration. If the graph is linear as in Figure 2 for a standard T-stage MSP then Algorithm 1 is equivalent to the original SDDP algorithm of [41] with one forward pass per iteration. The extension to general graphs was developed in [13].

The while loop yields a forward pass, simulating the performance of the policy defined by iteratively solving $\mathbf{SP}_i^{K-1}(\bar{x}, \omega)$ over a sample path defined by nodes $i \in \mathcal{N}$ (sampled from Φ) and realizations ω (sampled from ω_i's pmf). The *rand*() function gives a uniform $(0, 1)$ random variable and accounts for the implicit termination of the absorbing Markov chain. By performing multiple forward passes with a fixed set of cuts for each $\mathbf{SP}_i^{K-1}$ we can form a sample mean estimator of that policy's expected cost.

In Algorithm 1's backward pass, we iterate through list $\mathcal{S}$, from the last element back to the first, via *reverse*($\mathcal{S}$), i.e., reversing the sample path. The backward pass *trains the policy*, i.e., refines the set of cuts that approximate the expected cost-to-go function at nodes encountered on the forward pass. In the pseudo-code, we append an iteration index K to $\bar{x}$, the objective function value, and dual variable λ to emphasize the iteration dependence in each cut.

Algorithm 1: K^{th} iteration of SDDP under recursion (2).

```
/* Forward pass                                                          */
set x̄ = x_R, S = [ ], i = R
while rand() < Σ_{j∈i⁺} φ_ij do
    sample new i from i⁺ according to pmf, φ_{i,·} / Σ_{j∈i⁺} φ_ij
    sample ω from Ω_i according to pmf ℙ_i
    solve SP_i^{K-1}(x̄, ω) and obtain x'
    set x̄ = x'
    append (i, x̄) to the list S
end
/* Backward pass                                                         */
for (i, x̄^K) in reverse(S) do
    if i⁺ = ∅ then
        continue to next for-loop iterate of reverse(S)
    end
    for j ∈ i⁺, ω ∈ Ω_j do
        solve SP_j^{K-1}(x̄^K, ω)
        set V_j^{K-1}(x̄^K, ω) to the optimal objective function value
        set λ̄^K_{j,ω} to an optimal dual variable λ
    end
    set β_{i,K} = 𝔼_{j∈i⁺; ω_j} [ λ̄^K_{j,ω_j} ]
    set α_{i,K} = 𝔼_{j∈i⁺; ω_j} [ V_j^{K-1}(x̄^K, ω_j) ] − β_{i,K}^⊤ x̄^K
    add the cut θ ≥ α_{i,K} + β_{i,K}^⊤ x' to create SP_i^K
end
```

6.2 SDDP for decision-dependent transitions

Section 3.1 develops two models that allow Φ to depend on the agent's actions, yielding non-convex functions $V_i(x, \omega_i)$. Here we derive novel convex relaxations to enable computation via an SDDP variant. In SDDP's backward pass, we use these relaxations to compute cuts at each node. In SDDP's forward pass we use the *non-convex* models, coupled with these policy-defining cuts, to form a sample mean upper-bound estimator. Our approximations are relaxations, so we also obtain a lower bound on the model's optimal value, giving a posterior statistical bound on the policy's optimality gap.

6.2.1 An adaptive convex relaxation

We equivalently reformulate recursion (4), pulling decisions y back one time period, and carrying y into node i in the state. There are two implications, which exploit the fact that in (4) we can restrict attention to polytope $\mathcal{Y}$'s extreme points (see Section 3.1), which we denote $\text{ext}(\mathcal{Y}) = \{y_e : e \in \mathcal{E}\}$, and assume to be modest in number. The first is that the initial y decision is moved to the root. This is easily handled by making, or enumerating, those decisions at the root. The second is that, rather than a single decision y at node i, we have one for each child $j \in i^+$. The reformulated recursion includes fishing constraints, details $(u, x, x') \in \mathcal{X}_i(\omega_i)$ by separating the control and transition constraints, expands V_i's arguments to include y, and restricts attention to extreme points of $\mathcal{Y}$:

$$
\begin{aligned}
V_i(\bar{x}, \bar{y}, \omega_i) = & \\
\min_{u,x,x',y,y'} \quad & C_i(u, x', \omega_i) + C_i'(y) + \sum_{j \in i^+} \phi_{ij}(y) \sum_{\omega \in \Omega_j} \mathbb{P}_j(\omega) \cdot V_j(x', y_j', \omega) \\
\text{s.t.} \quad & u \in U_i(x, \omega_i) \\
& x' = T_i(u, x, \omega_i) \\
& y_j' \in \text{ext}(\mathcal{Y}), \ j \in i^+ \\
& x = \bar{x} \\
& y = \bar{y}.
\end{aligned} \tag{13}
$$

We simplify (A4) to the case of linear programming, and we revise (A5):

(A4') Problem (13) is a linear program in (u, x, x', y, y') when removing the cost-to-go function; $\Phi_0(\cdot)$ satisfies (5); and $\mathcal{Y}$ is a polytope.

(A5') For any sequence of $y \in \text{ext}(\mathcal{Y})$ the one-step matrices $\Phi_0(\cdot)$ are such that either the policy graph is acyclic or the Bellman operator for problem (13) is a γ-contraction.

Here, (A4') relies on the equivalence of (13) under $y_j' \in \mathcal{Y}$ without the cost-to-go function. We derive a convex approximation of $V_i(\bar{x}, \bar{y}, \omega_i)$, and compute cuts for SDDP using that approximation. We inductively assume:

$$
\sum_{\omega \in \Omega_j} \mathbb{P}_j(\omega) \cdot V_j(x', y_j', \omega) \geq \max_{k \in \mathcal{K}} \left[\alpha_{j,k} + \beta_{j,k}^\top x' + \gamma_{j,k}^\top y_j' \right], \tag{14}
$$

for $j \in i^+$ where we assume $\mathcal{K} = \{0, 1, 2, \ldots, K\}$ after K iterations of SDDP, and we suppress a j index on $\mathcal{K}$. Using compactness from (A3) and (A4'), expression (14) holds for $K = 0$ with $\beta_{j,0} = \gamma_{j,0} = 0$ and with $\alpha_{j,0} = -M$ for M sufficiently large. We show that the form of (14) persists as we compute cuts, reversing the sample path from a forward pass, as in Algorithm 1. Our convex lower-bounding approximation is as follows:

$V_i(\bar{x}, \bar{y}, \omega_i) \geq V_i^K(\bar{x}, \bar{y}, \omega_i) =$

$$
\begin{aligned}
\mathbf{SP}_i^K : \min_{u,x,x',y,y',\theta} \quad & C_i(u, x', \omega_i) + C_i'(y) + \sum_{j \in i^+} \theta_j \\
\text{s.t.} \quad & u \in U_i(x, \omega_i) \\
& x' = T_i(u, x, \omega_i) \\
& \theta_j \geq \phi_{ij}(y) \left(\alpha_{j,k} + \beta_{j,k}^\top x' + \gamma_{j,k}^\top y_j' \right), \quad j \in i^+, k \in \mathcal{K} \\
& y_j' \in \text{ext}(\mathcal{Y}), \ j \in i^+ \\
& x = \bar{x} \\
& y = \bar{y}
\end{aligned} \tag{15}
$$

$\geq \underline{V}_i^K(\bar{x}, \bar{y}, \omega_i) =$

$$
\begin{aligned}
\underline{\mathbf{SP}}_i^K : \min_{u,x,x',y,y',\theta} \quad & C_i\left(u, \sum_{e \in \mathcal{E}} x_e', \omega_i\right) + C_i'(y) + \sum_{j \in i^+} \theta_j \\
\text{s.t.} \quad & u \in U_i(x, \omega_i) \\
& \sum_{e \in \mathcal{E}} x_e' = T_i(u, x, \omega_i) \\
& \theta_j - \sum_{e \in \mathcal{E}} \phi_{ij}(y_e) \left(\beta_{j,k}^\top x_e' + \gamma_{j,k}^\top y_{j,e}' \right) \geq \phi_{ij}(y) \cdot \alpha_{j,k}, \quad j \in i^+, k \in \mathcal{K}, \\
& \sum_{e \in \mathcal{E}} y_{j,e}' \in \mathcal{Y}, \ j \in i^+ \\
& x = \bar{x} \quad [\lambda_x] \\
& y = \bar{y} \quad [\lambda_y].
\end{aligned} \tag{16}
$$

The first inequality holds by (14). Decision $\bar{y}$ computed in a forward pass of SDDP is an extreme point of $\mathcal{Y}$, say $\bar{y} = y_{\bar{e}}$. The inequality between (15) and (16) holds because the restriction $x_e' = 0$ and $y_{j,e}' = 0$ for $e \in \mathcal{E} \setminus \{\bar{e}\}$ in the latter yields the former, modulo the relaxation from constraints involving $\text{ext}(\mathcal{Y})$ to those with $\mathcal{Y}$. Under (A4'), model (16) is a linear program, parameterized in its right-hand side by $\bar{x}$ and $\bar{y}$. Thus, $\underline{V}_i^K(\cdot, \cdot, \omega_i)$ is convex.

In extending Algorithm 1's logic, assume that we are computing the K-th cut *from* node j with incoming state $(\bar{x}^K, \bar{y}^K)$. Then from subproblem (16), with i replaced by j, we obtain $\bar{\lambda}_{x,j,\omega}^K$ and $\bar{\lambda}_{y,j,\omega}^K$ and compute cuts as follows:

$$
\begin{aligned}
\beta_{j,K} &= \mathbb{E}_{\omega_j}\left[\bar{\lambda}_{x,j,\omega_j}^K\right] && \text{(17a)} \\
\gamma_{j,K} &= \mathbb{E}_{\omega_j}\left[\bar{\lambda}_{y,j,\omega_j}^K\right] && \text{(17b)} \\
\alpha_{j,K} &= \mathbb{E}_{\omega_j}\left[\underline{V}_j^K(\bar{x}^K, \bar{y}^K, \omega_j)\right] - \beta_{j,K}^\top \bar{x}^K - \gamma_{j,K}^\top \bar{y}^K. && \text{(17c)}
\end{aligned}
$$

The cuts are an outer linearization of $\mathbb{E}_{\omega_j}\left[\underline{V}_j^{K-1}(x, y, \omega_j)\right]$. We summarize our derivation in the following result.

Theorem 1. *Let (A1), (A2), (A3), (A4'), and (A5') hold, assume y is an extreme point of polytope $\mathcal{Y}$, and assume inequality (14) holds. Then*

$$\underline{V}_i^K(x, y, \omega_i) \leq V_i^K(x, y, \omega_i) \leq V_i(x, y, \omega_i),$$

where these are the respective optimal values of (16), (15), and (13). We have

$$\underline{V}_i^{K-1}(x, y, \omega_i) \leq \underline{V}_i^K(x, y, \omega_i) \text{ and } V_i^{K-1}(x, y, \omega_i) \leq V_i^K(x, y, \omega_i).$$

Moreover, $\underline{V}_i^K(\cdot, \cdot, \omega_i)$ is convex, and

$$\mathbb{E}_{\omega_i} \underline{V}_i^K(x, y, \omega_i) \geq \max_{k \in \mathcal{K}} \left[\alpha_{i,k} + \beta_{i,k}^\top x + \gamma_{i,k}^\top y\right],$$

where $\alpha_{i,k}, \beta_{i,k}, \gamma_{i,k}$ are computed by equation (17) for $i \in \mathcal{N}$, $k \in \mathcal{K}$.

Algorithm 2 gives the K-th iteration of our SDDP variant. We solve non-convex subproblems $\mathbf{SP}_i^{K-1}$ in forward passes, and solve $\underline{\mathbf{SP}}_i^{K-1}$ in backward passes. In recursion (4), we can equivalently replace $\mathcal{Y}$ with $\text{ext}(\mathcal{Y})$, but not in (15) and (16). Solving $\mathbf{SP}_i^{K-1}$ requires binary variables to handle $y'_j \in \text{ext}(\mathcal{Y})$ and linearizations for bilinear terms involving binaries. Subproblem $\underline{\mathbf{SP}}_i^{K-1}$ is a linear program. Coupled with (14) holding when $K = 0$, Theorem 1 shows that Algorithm 2's cuts form an outer approximation at each node. In general, we obtain a suboptimal policy because of the problem's non-convexity, but we can assess an optimality gap using bounds as discussed above.

Algorithm 2: K^{th} iteration of SDDP under recursion (4).

```
/* Forward pass                                                              */
set x̄ = x_R, S = [ ], i = R, ȳ = 0 and ȳ' ∈ Y
while rand() < Σ_{j∈i+} φ_ij(ȳ) do
    sample new i from i+ according to pmf, φ_{i,·}(ȳ) / Σ_{j∈i+} φ_ij(ȳ)
    sample ω from Ω_i according to pmf P_i
    solve SP_i^{K-1}(x̄, ȳ', ω) and obtain x' and y' = [y'_j]_{j∈i+}
    append (i, x', y') to the list S
    set x̄ = x', ȳ = ȳ', ȳ' = y'
end
/* Backward pass                                                             */
for (i, x̄^K, ȳ^K) in reverse(S) do
    if i+ = ∅ then
        continue to next for-loop iterate of reverse(S)
    end
    for j ∈ i+ do
        for ω ∈ Ω_j do
            solve SP_j^{K-1}(x̄^K, ȳ_j^K, ω)
            set V_j^{K-1}(x̄^K, ȳ_j^K, ω) to the optimal objective function value
            set λ̄^K_{x,j,ω} and λ̄^K_{y,j,ω} to optimal dual variables λ_x and λ_y
        end
        compute cut coefficients α_{j,K}, β_{j,K}, γ_{j,K} using equation (17)
        add the K^{th} cut to SP_i^K and to SP_i^K using α_{j,K}, β_{j,K}, γ_{j,K}
    end
end
```

6.2.2 Lagrangian relaxation

We succinctly repeat the analysis of Section 6.2.1 but starting from recursion (6), seeking a convex relaxation for $V_i(\bar{x}, \omega_i)$. We inductively assume for $j \in i^+$ and $\omega \in \Omega_j$ that

$$V_j(x', \omega) \geq \max_{k \in \mathcal{K}} \left[\alpha_{j,\omega,k} + \beta_{j,\omega,k}^\top x' \right], \tag{18}$$

which holds at $K = 0$ with $\beta_{j,\omega,0} = 0$ and $\alpha_{j,\omega,0} = -M$ for M sufficiently large. Applying (18) in recursion (6), and adding a fishing constraint $x = \bar{x}$ yields the following analog of subproblem (15):

$$\begin{aligned}
\mathbf{SP}_i^K : V_i^K(\bar{x}, \quad & \omega_i) = \\
\min_{u,x,x',y,\theta,\Theta} \quad & C_i(u, x', y, \omega_i) + \Theta \\
\text{s.t.} \quad & (u, x, x', y) \in \mathcal{X}_i(\omega_i) \\
& \sum_{d \in D} y_d = 1 \\
& y_d \in \{0, 1\}, \quad d \in D \\
& \Theta \geq \sum_{d \in D} y_d \sum_{j \in i^+;\ \omega \in \Omega_j} \phi_{ij}^d \cdot \mathbb{P}_j(\omega) \cdot \theta_{j,\omega} \\
& \theta_{j,\omega} \geq \alpha_{j,\omega,k} + \beta_{j,\omega,k}^\top x', \ j \in i^+, \omega \in \Omega_j, k \in \mathcal{K} \\
& x = \bar{x} \quad [\lambda].
\end{aligned} \tag{19}$$

The constraint for Θ is nonlinear because binary variable y_d multiplies continuous variable $\theta_{j,\omega}$, but these can again be equivalently linearized. If C_i and $\mathcal{X}_i$ permit a linear formulation, then (19) is a mixed-integer linear program, and $V_i^K(\cdot, \omega_i)$ is non-convex. Because $\mathbf{SP}_i^K$ includes integer variables, we do not have

immediate access to a subgradient λ. We could relax integrality when solving $\mathbf{SP}_k^{K-1}$ on the backward pass. Importantly, we would enforce integrality when solving $\mathbf{SP}_k^{K-1}$ on the forward pass. Instead, we take the Lagrangian dual of the $x = \bar{x}$ to obtain the following analog of (16):

$$\begin{array}{rll} \underline{\mathbf{SP}}_i^K : \underline{V}_i^K(\bar{x}, & \omega_i) = & \\ \max_{\lambda} \min_{u,x,x',y,\theta,\Theta} & C_i(u,x',y,\omega_i) + \Theta - \lambda^\top (x - \bar{x}) & \\ \text{s.t.} & (u,x,x',y) \in \mathcal{X}_i(\omega_i) & \\ & \sum_{d \in D} y_d = 1 & \\ & y_d \in \{0,1\}, \quad d \in D & \\ & \Theta \geq \sum_{d \in D} y_d \sum_{j \in i^+;\ \omega \in \Omega_j} \phi_{ij}^d \cdot \mathbb{P}_j(\omega) \cdot \theta_{j,\omega} & \\ & \theta_{j,\omega} \geq \alpha_{j,\omega,k} + \beta_{j,\omega,k}^\top x',\ j \in i^+, \omega \in \Omega_j, k \in \mathcal{K}. & \end{array} \tag{20}$$

We do not state the algorithm associated with (19) and (20) because of its similarity to Algorithm 2, but we make some comments. With notation parallel to that of Algorithms 1 and 2 the cuts we compute have form:

$$\theta_{j,\omega} \geq \underline{V}_j^K(\bar{x}^K, \omega) + \left(\bar{\lambda}_{j,\omega}^K\right)^\top \left(x' - \bar{x}^K\right)$$

so that:

$$\beta_{j,\omega,K} = \bar{\lambda}_{j,\omega}^K \tag{21a}$$

$$\alpha_{j,\omega,K} = \underline{V}_j^K(\bar{x}^K, \omega) - \left(\bar{\lambda}_{j,\omega}^K\right)^\top \bar{x}^K. \tag{21b}$$

We solve the non-convex problem (19) on the forward pass, and solve the convex relaxation (20) on the backward pass to compute cuts.

Solving the convex relaxation (20) can be expensive because it requires solving a set of mixed-integer programs at each node for the Lagrangian dual, e.g., using a cutting-plane or subgradient method. That said, we can compute valid but possibly weaker cuts with a few iterations of those methods. We summarize our convex relaxation in the following result.

Theorem 2. *Let (A1), (A2), (A3), and (A5') hold, and assume inequality (18) holds. Then*

$$\underline{V}_i^K(x, \omega_i) \leq V_i^K(x, \omega_i) \leq V_i(x, \omega_i),$$

where these are the respective optimal values of (20), (19), and (6). We have

$$\underline{V}_i^{K-1}(x, \omega_i) \leq \underline{V}_i^K(x, \omega_i) \text{ and } V_i^{K-1}(x, \omega_i) \leq V_i^K(x, \omega_i).$$

Moreover, $\underline{V}_i^K(\cdot, \omega)$ is convex, and

$$\underline{V}_i^K(x, \omega) \geq \max_{k \in \mathcal{K}} \left[\alpha_{i,\omega,k} + \beta_{i,\omega,k}^\top x\right],$$

where $\alpha_{i,\omega,k}, \beta_{i,\omega,k}$ are computed by equation (21) for $i \in \mathcal{N}$, $k \in \mathcal{K}$.

In (A5') we use $\Phi(y) = \sum_{d \in D} y_d \Phi^d$, and (A4) is not required because the Lagrangian dual yields a convex relaxation even though the inner minimization in (20) is non-convex. For this reason, the method could handle additional discreteness or non-convexity in $(u, x, x', y) \in \mathcal{X}_i(\omega_i)$.

6.3 SDDP for decision-dependent learning

We parallel Section 6.2.1's development, although we could use Section 6.2.2. We reformulate (11) to give the analog of (13), accounting for the belief pmf:

$$
\begin{aligned}
V_i(\bar{x}, \bar{y}, b, \omega_i) = & \\
\min_{u,x,x',y,y'} \quad & C_i(u, x', \omega_i) + C_i'(y) + \sum_{\substack{m \in \mathcal{M} \\ j \in i^+ \\ \omega \in \Omega_j}} b_m \cdot \phi_{ij}^m(y) \cdot \mathbb{P}_j^m(\omega) \cdot V_j(x', y_j', b', \omega) \\
\text{s.t.} \quad & u \in U_i(x, \omega_i) \\
& x' = T_i(u, x, \omega_i) \\
& y_j' \in \text{ext}(\mathcal{Y}), \ j \in i^+ \\
& x = \bar{x} \\
& y = \bar{y},
\end{aligned} \tag{22}
$$

where $b' = B(y, b, i \to j, \omega)$. We adapt our assumptions, accounting for both decision-dependence and learning:

(A4") Problem (22) is a linear program in (u, x, x', y, y') when removing the cost-to-go function; $\Phi_0^m(\cdot)$ satisfies (5) for all $m \in \mathcal{M}$; and $\mathcal{Y}$ is a polytope.

(A5") For any sequence of $y \in \text{ext}(\mathcal{Y})$ and for all $m \in \mathcal{M}$ the one-step matrices $\Phi_0^m(\cdot)$ are such that either the policy graph is acyclic or the Bellman operator for problem (22) is a γ-contraction.

We assume:

$$
\sum_{\omega \in \Omega_j} \mathbb{P}_j^m(\omega) \cdot V_j(x', y_j', b', \omega) \geq \max_{k \in \mathcal{K}} \left[\alpha_{j,m,k} + \beta_{j,m,k}^\top x' + \gamma_{j,m,k}^\top y_j' \right]. \tag{23}
$$

Then we can interpolate cuts for different belief states as in [15] to form:

$$
\begin{aligned}
V_i^K(\bar{x}, \bar{y}, b, \omega_i) = & \\
\min_{u,x,x',y,y',\theta} \max_{\eta \geq 0} \quad & C_i(u, x', \omega_i) + C_i'(y) + \sum_{k \in \mathcal{K}} \eta_k \theta_k \\
\text{s.t.} \quad & u \in U_i(x, \omega_i) \\
& x' = T_i(u, x, \omega_i) \\
& \theta_k \geq \sum_{\substack{m \in \mathcal{M} \\ j \in i^+}} b_{m,k} \cdot \phi_{ij}^m(y) \left(\alpha_{j,m,k} + \beta_{j,m,k}^\top x' + \gamma_{j,m,k}^\top y_j' \right), \quad k \in \mathcal{K} \\
& \sum_{k \in \mathcal{K}} \eta_k b_k = b, \qquad [\mu] \\
& \sum_{k \in \mathcal{K}} \eta_k = 1, \qquad [\Theta] \\
& y_j' \in \text{ext}(\mathcal{Y}), \ j \in i^+ \\
& x = \bar{x} \quad [\lambda_x] \\
& y = \bar{y} \quad [\lambda_y].
\end{aligned}
$$

Taking the dual of the inner maximization, we obtain the analog of (15):

$$
\begin{aligned}
&\mathbf{SP}_i^K: \\
&V_i^K(\bar{x},\bar{y},b,\omega_i) = \\
&\qquad \min_{u,x,x',y,y',\theta,\mu,\Theta} \quad C_i(u,x',\omega_i) + C_i'(y) + b^\top \mu + \Theta \\
&\qquad \text{s.t.} \quad u \in U_i(x,\omega_i) \\
&\qquad\qquad x' = T_i(u,x,\omega_i) \\
&\qquad\qquad b_k^\top \mu + \Theta \geq \theta_k, \quad k \in \mathcal{K} \\
&\qquad\qquad \theta_k \geq \sum_{\substack{m\in\mathcal{M}\\ j\in i^+}} b_{m,k}\cdot\phi_{ij}^m(y)\left(\alpha_{j,m,k} + \beta_{j,m,k}^\top x' + \gamma_{j,m,k}^\top y_j'\right), \quad k\in\mathcal{K} \\
&\qquad\qquad y_j' \in \mathrm{ext}(\mathcal{Y}),\ j\in i^+ \\
&\qquad\qquad x = \bar{x} \quad [\lambda_x] \\
&\qquad\qquad y = \bar{y} \quad [\lambda_y].
\end{aligned}
\tag{24}
$$

Model (24) is non-convex but can be solved as a mixed-integer linear program on forward passes of SDDP. Relaxing in the same way as in Section 6.2.1 yields the analog of (16):

$$
\begin{aligned}
&\underline{\mathbf{SP}}_i^K: \\
&\underline{V}_i^K(\bar{x},\bar{y},b,\omega_i) = \\
&\qquad \min_{u,x,x',y,y',\theta,\theta',\mu,\Theta} \quad C_i(u,x',\omega_i) + C_i'(y) + b^\top \mu + \Theta \\
&\qquad \text{s.t.} \quad u \in U_i(x,\omega_i) \\
&\qquad\qquad x' = T_i(u,x,\omega_i) \\
&\qquad\qquad b_k^\top \mu + \Theta \geq \theta_k, \quad k \in \mathcal{K} \\
&\qquad\qquad \theta_k - \sum_{\substack{e\in\mathcal{E}\\ m\in\mathcal{M}\\ j\in i^+}} b_{m,k}\cdot\phi_{ij}^m(y_e)\cdot\theta_{j,m,e,k}' \geq \sum_{\substack{m\in\mathcal{M}\\ j\in i^+}} b_{m,k}\cdot\phi_{ij}^m(y)\cdot\alpha_{j,m,k}, k\in\mathcal{K} \\
&\qquad\qquad \theta_{j,m,e,k}' \geq \beta_{j,m,k}^\top x_e' + \gamma_{j,m,k}^\top y_{j,e}', \quad j\in i^+, m\in\mathcal{M}, e\in\mathcal{E}, k\in\mathcal{K} \\
&\qquad\qquad x' = \sum_{e\in\mathcal{E}} x_e' \\
&\qquad\qquad \sum_{e\in\mathcal{E}} y_{j,e}' \in \mathcal{Y},\ j\in i^+ \\
&\qquad\qquad x = \bar{x} \quad [\lambda_x] \\
&\qquad\qquad y = \bar{y} \quad [\lambda_y].
\end{aligned}
\tag{25}
$$

Under (A4"), model (25) is a linear program, which is parameterized in its right-hand side by $(\bar{x},\bar{y})$ and its objective function by b. Thus, $\underline{V}_i^K(\bar{x},\bar{y},b,\omega_i)$ is convex in $(\bar{x},\bar{y})$ for fixed b and concave in b for fixed $(\bar{x},\bar{y})$.

With $\mathbb{E}_{\omega_j^m}[\cdot]$ denoting $\sum_{\omega\in\Omega_j}\mathbb{P}_j^m(\omega)[\cdot]$ we compute cut coefficients:

$$
\begin{aligned}
\beta_{j,m,K} &= \mathbb{E}_{\omega_j^m}\left[\bar{\lambda}_{x,j,\omega_j^m}^K\right] && \text{(26a)}\\
\gamma_{j,m,K} &= \mathbb{E}_{\omega_j^m}\left[\bar{\lambda}_{y,j,\omega_j^m}^K\right] && \text{(26b)}\\
\alpha_{j,m,K} &= \mathbb{E}_{\omega_j^m}\left[\underline{V}_j^K(\bar{x}^K,\bar{y}^K,b^K,\omega_j^m)\right] - \beta_{j,m,K}^\top \bar{x}^K - \gamma_{j,m,K}^\top \bar{y}^K. && \text{(26c)}
\end{aligned}
$$

The cuts are an outer linearization of

$$\sum_{\omega\in\Omega_j}\mathbb{P}_j^m(\omega)\cdot\underline{V}_j^K(x',y',b',\omega).$$

The above development yields the following result.

Theorem 3. *Let (A1), (A2), (A3), (A4"), and (A5") hold, assume y is an extreme point of polytope $\mathcal{Y}$, and assume inequality (23) holds. Then*

$$\underline{V}_i^K(x, y, b, \omega_i) \le V_i^K(x, y, b, \omega_i) \le V_i(x, y, b, \omega_i),$$

where these are the respective optimal values of (25), (24), and (22). We have

$$\underline{V}_i^{K-1}(x, y, b, \omega_i) \le \underline{V}_i^K(x, y, b, \omega_i) \text{ and } V_i^{K-1}(x, y, b, \omega_i) \le V_i^K(x, y, b, \omega_i).$$

Moreover, $\underline{V}_i^K(x, y, b, \omega_i)$ is convex in (x, y) for fixed b and concave in b for fixed (x, y). And,

$$\mathbb{E}_{\omega_i^m} \underline{V}_i^K(x, y, b, \omega_i) \ge \max_{k \in \mathcal{K}} \left[\alpha_{i,m,k} + \beta_{i,m,k}^\top x + \gamma_{i,m,k}^\top y\right],$$

where $\alpha_{i,m,k}, \beta_{i,m,k}, \gamma_{i,m,k}$ are computed by (26) for $i \in \mathcal{N}$, $m \in \mathcal{M}$, $k \in \mathcal{K}$.

Algorithm 3 gives the K-th iteration of our variant of SDDP. We solve non-convex subproblems $\mathbf{SP}_i^{K-1}$ in forward passes and linear programs $\underline{\mathbf{SP}}_i^{K-1}$ in backward passes. The forward pass accounts for our belief state, and its updates, when sampling and when solving $\mathbf{SP}_i^{K-1}$. In the backward pass subproblems $\underline{\mathbf{SP}}_j^{K-1}$ are identical for each model m, and only when computing cuts must we account for $\mathbb{P}_j^m$ via equation (26). We include b^K when adding cuts to $\mathbf{SP}_i^K$ and $\underline{\mathbf{SP}}_i^K$ because of the $b_{m,k}$ terms in the resulting inequalities.

Algorithm 3: K^{th} iteration of SDDP under recursion (9).

```
/* Forward pass                                                              */
set x̄ = x_R, b = b_R, S = [ ], i = R, ȳ = 0, and ȳ' ∈ Y
while rand() < Σ_{m∈M, j∈i+} b_m φ_ij^m(ȳ) do
    set i⁻ = i
    sample new i from i⁺ according to pmf, (Σ_{m∈M} b_m φ_{i,·}^m(ȳ)) / (Σ_{m∈M, j∈i+} b_m φ_ij^m(ȳ))
    sample ω from Ω_i according to pmf mixture Σ_{m∈M} b_m ℙ_i^m
    set b' = B(ȳ, b, i⁻ → i, ω) using (12)
    solve SP_i^{K-1}(x̄, ȳ', b', ω) and obtain x' and y' = [y'_j]_{j∈i+}
    append (i, x', y', b') to the list S
    set x̄ = x', ȳ = ȳ', ȳ' = y', b = b'
end
/* Backward pass                                                             */
for (i, x̄^K, ȳ^K, b^K) in reverse(S) do
    if i⁺ = ∅ then
        continue to next for-loop iterate of reverse(S)
    end
    for j ∈ i⁺ do
        for ω ∈ Ω_j do
            solve SP_j^{K-1}(x̄^K, ȳ_j^K, b^K, ω)
            set V_j^{K-1}(x̄^K, ȳ_j^K, b^K, ω) to the optimal objective function value
            set λ̄^K_{x,j,ω} and λ̄^K_{y,j,ω} to optimal dual variables λ_x and λ_y
        end
        for m ∈ M do
            compute cut coefficients α_{j,m,K}, β_{j,m,K}, γ_{j,m,K} using equation (26)
            add K^{th} cut to SP_i^K and SP_i^K using α_{j,m,K}, β_{j,m,K}, γ_{j,m,K} and b^K
        end
    end
end
```

7 Computational examples

As a computational demonstration of our algorithms, we solve two examples in `SDDP.jl`, a free and open-source Julia [7] package for modeling policy graphs and solving them with stochastic dual dynamic programming [14]. `SDDP.jl` is based on the JuMP modeling language [37]. The code and data for our experiments are available at https://github.com/odow/SDDP.jl. Our intent is neither to demonstrate large-scale examples nor to implement the adaptive decision-dependent learning algorithm; we leave that to future work.

7.1 The cheese producer

To demonstrate the Lagrangian relaxation algorithm from Section 6.2.2, we solve Example 4. As data, we use $\rho = 0.9$, $c_y = 3$, $c_u = 1$, $\Omega_q = \{0, 2, 4, 6, 8\}$, $\Omega_d = \{5, 10\}$, each with a uniform pmf. One sample path of the policy is shown in Figure 13. The policy appears to correspond to the decision rule that $y = 1$ if $x' \geq 5$ and $y = 0$ if $x' < 5$.

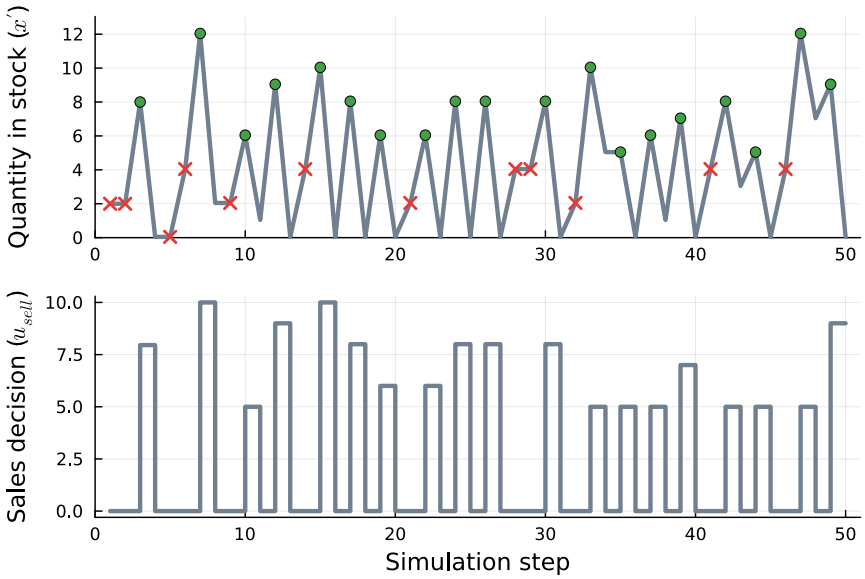

Figure 13: One simulation trajectory of the cheese producer policy over 50 time-steps. The red crosses correspond to states when $y = 0$ and the farmer does not visit the market, and the green circles correspond to states when $y = 1$ and the farmer visits the market.

We further analyze Example 4, reformulating it with a finite-horizon of $T = 2$ to 10 weeks. Each week consists of the farm node and the optional market node. We train each policy for 200 iterations of SDDP and then perform 1000 simulations of the policy. Figure 14's violin plots show the distribution of simulated objective values, where we maximize profit. Due to the non-convex value function, our upper bound does not converge to the sample mean. However, if we have small gaps, it gives us confidence that the policies are good.

7.2 The tiger problem

To demonstrate our decision-dependent statistical learning model, we solve Example 10 using Algorithm 3 but with the Lagrangian relaxation from Section 6.2.2. We use a discount factor of $\rho = 0.95$, and we use a false positive rate of 15%. Figure 15 shows 100 simulations of the belief that the tiger is behind the left door over the number of time steps, as well as the net count of the number of times we hear the tiger behind the left door. The policy corresponds to the decision rule of opening the opposite door once the net count reaches three. Of the 100 simulations, there is one example in which the agent opens the "wrong" door to the tiger after hearing three consecutive false positives.

We further analyze Example 10 via a parametric study for false positive rates of 0% (perfect hearing), 10%, 20%, and 30%. We train each policy for 100 iterations and then perform 100 simulations of the policy. Figure 16 shows violin plots of the simulated objective values. As expected, as the false positive rate grows, the agent has a harder time learning the location of the tiger, and the cost increases. Because of the non-convex cost-to-go function, our lower bound does not converge to the sample mean. However, if we have small gaps, then we have confidence that the policies are good in practice.

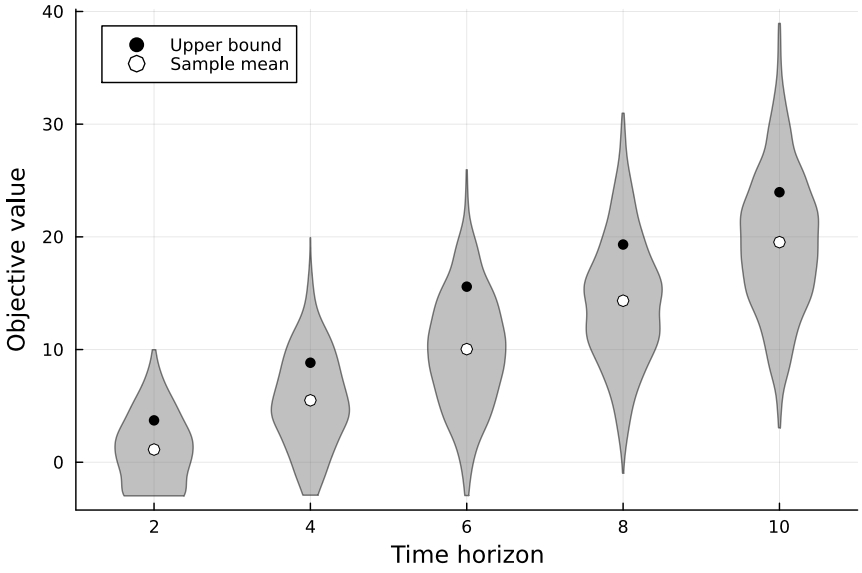

Figure 14: Violin plots of the distribution of 1000 simulated objective values for Example 4 with differing finite time horizon T. The black dot corresponds to the upper bound on expected profit obtained from SDDP. The white dot is the sample mean of the distribution.

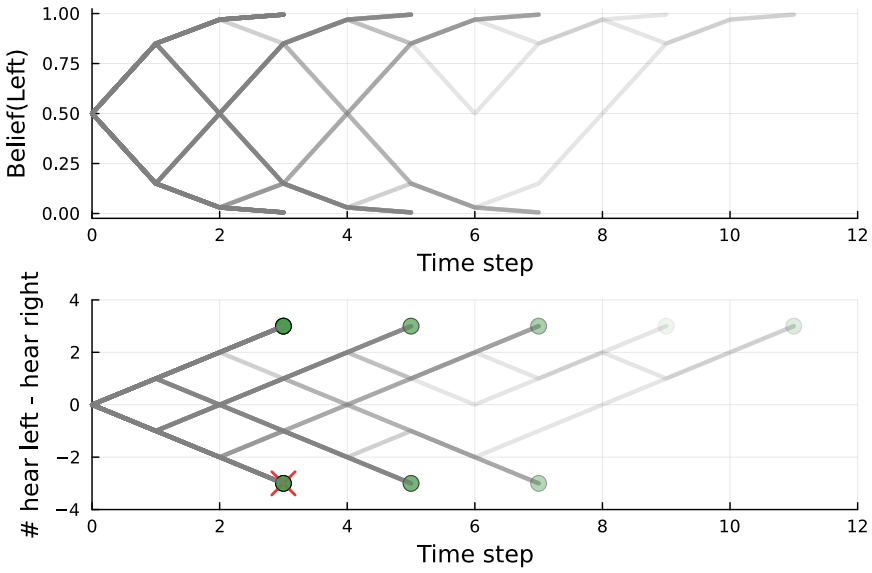

Figure 15: One hundred simulations of the tiger policy with a false positive rate of 15%. The green circles correspond to "correctly" opening the door to escape. There is one red cross at position $(3, -3)$, which is a simulation that "wrongly" opened the door to the tiger after hearing three consecutive false positives.

8 Conclusion

We have shown how policy graphs can model a range of sequential decision problems under uncertainty. From the viewpoint of stochastic programming, we have extended policy graphs to incorporate ideas from MDPs, including decision-dependent transition probabilities, a limited form of statistical learning, and their combination: decision-dependent learning. From the viewpoint of the MDP literature, we have presented policy graphs and SDDP algorithms as a practical combination for modeling and solving a class of structured

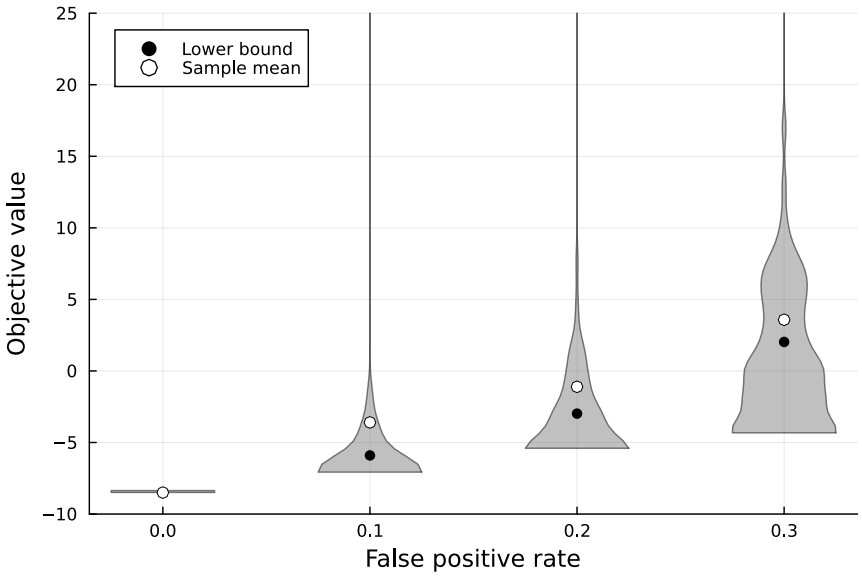

Figure 16: Violin plots of the distribution of 100 simulated objective values for the tiger problem with varying false positive rates. The black dot corresponds to the lower bound of the expected value obtained from SDDP. The white dot is the sample mean of the distribution. The thin upper tails (full extent not shown) represent simulations in which the door to the tiger was opened and a cost of $100 was incurred.

MDPs with continuous states and continuous actions, which we call CACS MDPs.

Our model for decision-dependent learning allows the agent to take actions that reveal information so that better decisions can be subsequently made. Our models for both decision-dependent transitions and decision-dependent learning yield non-convex cost-to-go functions. Our algorithmic variants of SDDP use a mix of convex relaxations to form a policy, and the original non-convex models to evaluate the policy's performance.

Although our focus has been on modeling and algorithmic development, we provide an open-source implementation of our ideas in `SDDP.jl` [14]. The code and documentation are freely available at https://sddp.dev. As future work, we will conduct larger-scale computational experiments and integrate our work with other features supported by `SDDP.jl`, such as risk aversion.

Acknowledgments

The authors thank two anonymous referees for comments that improved the paper. David Morton's research was supported, in part, by the US Department of Homeland Security (DHS) under Grant Award Number 17STQAC00001-06-00. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DHS.

References

[1] Adiga, R.: Optimizing geothermal well planning under reservoir uncertainty with stochastic programming. Ph.D. thesis, University of Auckland (2024)

[2] Ahmed, S.: Strategic planning under uncertainty: Stochastic integer programming approaches. Ph.D. thesis, University of Illinois at Urbana-Champaign (2000)

[3] Arslan, N., Dowson, O., Morton, D.P.: An SDDP algorithm for multistage stochastic programs with decision-dependent uncertainty. In: 2024 Winter Simulation Conference, pp. 3288–3299 (2024)

[4] Barto, A.G., Thomas, P.S., Sutton, R.S.: Some recent applications of reinforcement learning. In: Proceedings of the Eighteenth Yale Workshop on Adaptive and Learning Systems (2017)

[5] Basciftci, B., Ahmed, S., Gebraeel, N.: Adaptive two-stage stochastic programming with an analysis on capacity expansion planning problem. Manufacturing & Service Operations Management **26**(6), 2121–2141 (2024)

[6] Bellman, R.: Dynamic programming. Princeton University Press, Princeton, NJ (1957)

[7] Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM Review **59**(1), 65–98 (2017)

[8] Bielecki, T.R., Cialenco, I., Ruszczyński, A.: Risk filtering and risk-averse control of Markovian systems subject to model uncertainty. Mathematical Methods of Operations Research **98**(2), 231–268 (2023)

[9] Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Science & Business Media, New York, NY (2011)

[10] Bissiri, P.G., Holmes, C.C., Walker, S.G.: A general framework for updating belief distributions. Journal of the Royal Statistical Society Series B: Statistical Methodology **78**(5), 1103–1130 (2016)

[11] Chang, H.S., Hu, J., Fu, M.C., Marcus, S.I.: Simulation-based algorithms for Markov decision processes. Springer, London (2013)

[12] de Farias, D.P., van Roy, B.: The linear programming approach to approximate dynamic programming. Operations Research **51**(6), 850–865 (2003)

[13] Dowson, O.: The policy graph decomposition of multistage stochastic programming problems. Networks **76**(1), 3–23 (2020)

[14] Dowson, O., Kapelevich, L.: SDDP.jl: a Julia package for stochastic dual dynamic programming. INFORMS Journal on Computing **33**(1), 27–33 (2021)

[15] Dowson, O., Morton, D.P., Pagnoncelli, B.K.: Partially observable multistage stochastic programming. Operations Research Letters **48**(4), 505–512 (2020)

[16] Dupacová, J.: Optimization under exogenous and endogenous uncertainty. In: Proceedings of MME06, University of West Bohemia in Pilsen (2006)

[17] Elliott, R.J., Aggoun, L., Moore, J.B.: Hidden Markov models: estimation and control, vol. 29. Springer Science & Business Media, New York, NY (2008)

[18] Füllner, C., Rebennack, S.: Stochastic dual dynamic programming and its variants. SIAM Review **67**(3), 415–539 (2025)

[19] Ghatrani, Z., Ghate, A.: Inverse Markov decision processes with unknown transition probabilities. IISE Transactions **55**(6), 588–601 (2023)

[20] Girardeau, P., Leclère, V., Philpott, A.B.: On the convergence of decomposition methods for multistage stochastic convex programs. Mathematics of Operations Research **40**(1), 130–145 (2015)

[21] Goel, V., Grossmann, I.E.: A class of stochastic programs with decision dependent uncertainty. Mathematical Programming **108**(2), 355–394 (2006)

[22] Grünwald, P.: The safe Bayesian: learning the learning rate via the mixability gap. In: International Conference on Algorithmic Learning Theory, pp. 169–183. Springer (2012)

[23] Guigues, V.: Convergence analysis of sampling-based decomposition methods for risk-averse multistage stochastic convex programs. SIAM Journal on Optimization **26**(4), 2468–2494 (2016)

[24] Hallak, A., Di Castro, D., Mannor, S.: Contextual Markov decision processes. arXiv preprint arXiv:1502.02259 (2015)

[25] Hellemo, L., Barton, P.I., Tomasgard, A.: Decision-dependent probabilities in stochastic programs with recourse. Computational Management Science **15**, 369–395 (2018)

[26] Howard, R.: Dynamic programming and Markov processes. MIT Press, Cambridge, MA (1960)

[27] Ibrahim, J.G., Chen, M.H.: Power prior distributions for regression models. Statistical Science **16**(1), 46–60 (2000)

[28] Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101**, 99–134 (1998)

[29] King, A.J., Wallace, S.W.: Modeling with stochastic programming. Springer (2012)

[30] Lamas, P., Goycoolea, M., Pagnoncelli, B., Newman, A.: A target-time-windows technique for project scheduling under uncertainty. European Journal of Operational Research **314**(2), 792–806 (2024)

[31] Lan, G.: Policy mirror descent for reinforcement learning: Linear convergence, new sampling complexity, and generalized problem classes. Mathematical Programming **198**(1), 1059–1106 (2023)

[32] Lara, C.L., Mallapragada, D.S., Papageorgiou, D.J., Venkatesh, A., Grossmann, I.E.: Deterministic electric power infrastructure planning: Mixed-integer programming model and nested decomposition algorithm. European Journal of Operational Research **271**(3), 1037–1054 (2018)

[33] Lara, C.L., Siirola, J.D., Grossmann, I.E.: Electric power infrastructure planning under uncertainty: stochastic dual dynamic integer programming (SDDiP) and parallelization scheme. Optimization and Engineering **21**, 1243–1281 (2020)

[34] Le, T.P., Vien, N.A., Chung, T.: A deep hierarchical reinforcement learning algorithm in partially observable Markov decision processes. IEEE Access **6**, 49089–49102 (2018)

[35] Lejeune, M., Margot, F., de Oliveira, A.D.: Chance-constrained programming with decision-dependent uncertainty. In: Proc. Workshop New Directions Stochastic Optim. (2018)

[36] Li, Y.: Reinforcement learning applications. arXiv preprint arXiv:1908.06973 (2019)

[37] Lubin, M., Dowson, O., Dias Garcia, J., Huchette, J., Legat, B., Vielma, J.P.: JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. Mathematical Programming Computation **15**(3), 581-589 (2023)

[38] Mamon, R.S., Elliott, R.J.: Hidden Markov models in finance, vol. 4. Springer, New York, NY (2007)

[39] Maxwell, M.S., Restrepo, M., Henderson, S.G., Topaloglu, H.: Approximate dynamic programming for ambulance redeployment. INFORMS Journal on Computing **22**(2), 266–281 (2010)

[40] O'Hagan, A.: Fractional Bayes factors for model comparison. Journal of the Royal Statistical Society: Series B (Methodological) **57**(1), 99–118 (1995)

[41] Pereira, M.V.F., Pinto, L.M.V.G.: Multi-stage stochastic optimization applied to energy planning. Mathematical Programming **52**, 359–375 (1991)

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

[43] Powell, W.B.: Approximate dynamic programming: Solving the curses of dimensionality, 2nd ed edn. Wiley Series in Probability and Statistics. Wiley, Hoboken, N.J (2011)

[44] Powell, W.B.: Clearing the jungle of stochastic optimization. In: Bridging data and decisions, pp. 109–137. INFORMS (2014)

[45] Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, Hoboken, NJ (2014)

[46] Rosemberg, A., Street, A., Garcia, J.D., Silva, T., Valladão, D., Dowson, O.: Assessing the cost of network simplifications in hydrothermal power systems. IEEE Transactions on Sustainable Energy **13**(1), 196–206 (2022)

[47] Seranilla, B.K.: On the applications of stochastic dual dynamic programming. Ph.D. thesis, Université du Luxembourg, Luxembourg City, Luxembourg (2023)

[48] Shapiro, A., Dentcheva, D., Ruszczynski, A.: Lectures on stochastic programming: Modeling and theory. SIAM, Philadelphia, PA (2021)

[49] Siddig, M., Song, Y., Khademi, A.: Maximum-posterior evaluation for partially observable multistage stochastic programming. In: Proceedings of the 2021 IISE Annual Conference (2021)

[50] Steimle, L.N., Kaufman, D.L., Denton, B.T.: Multi-model Markov decision processes. IISE Transactions **53**(10), 1124–1139 (2021)

[51] Topaloglu, H., Powell, W.B.: Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. INFORMS Journal on Computing **18**(1), 31–42 (2006)

[52] Wang, K., Shah, S., Chen, H., Perrault, A., Doshi-Velez, F., Tambe, M.: Learning MDPs from features: Predict-then-optimize for sequential decision making by reinforcement learning. Advances in Neural Information Processing Systems **34**, 8795–8806 (2021)

[53] Wiesemann, W., Kuhn, D., Rustem, B.: Robust Markov decision processes. Mathematics of Operations Research **38**(1), 153–183 (2013)

[54] Yin, W., Li, Y., Hou, J., Miao, M., Hou, Y.: Coordinated planning of wind power generation and energy storage with decision-dependent uncertainty induced by spatial correlation. IEEE Systems Journal **17**(2), 2247–2258 (2022)

[55] Zou, J., Ahmed, S., Sun, X.A.: Stochastic dual dynamic integer programming. Mathematical Programming **175**(1-2), 461–502 (2019)