Symbolic Dynamic Programming for Continuous State and Action MDPs

Anonymous

Abstract

Many real-world decision-theoretic planning problems are naturally modeled using both continuous state and action (CSA) spaces, yet little work has provided *exact* solutions for the case of continuous actions. In this work, we propose a symbolic dynamic programming (SDP) solution to obtain the *optimal closed-form* value function and policy for CSA-MDPs with multivariate continuous state *and* actions, discrete noise, piecewise linear dynamics, and piecewise linear (or restricted piecewise quadratic) reward. Our key contribution over previous SDP work is to show how the continuous action maximization step in the dynamic programming backup can be evaluated optimally and symbolically — a task which amounts to *symbolic* constrained optimization subject to unknown state parameters; we further integrate this technique to work with an efficient and compact data structure for SDP — the extended algebraic decision diagram (XADD). We demonstrate empirical results on a didactic nonlinear planning example and two domains from operations research to show the *first automated exact solution* to these problems.

Introduction

Many real-world stochastic planning problems involving resources, time, or spatial configurations naturally use continuous variables in both their state and action representation. For example, in a MARS ROVER problem (Bresina et al. 2002), a rover must navigate within a continuous spatial environment and carry out assigned scientific discovery tasks; in INVENTORY CONTROL problems (Mahootchi 2009) for continuous resources such as petroleum products, a business must decide what quantity of each item to order subject to uncertain demand, (joint) capacity constraints, and reordering costs; and in RESERVOIR MANAGEMENT problems (Lamond and Boukhtouta 2002), a utility must manage continuous reservoir water levels in continuous time to avoid underflow while maximizing electricity generation revenue.

Previous work on *exact* solutions to multivariate continuous state *and* action settings has been quite limited. There are well-known exact solutions in the control theory literature for the case of linear-quadratic Gaussian (LQG) control (Athans 1971), i.e., minimizing a quadratic cost function subject to linear dynamics with Gaussian noise in a partially observed setting. However, the transition dynamics and reward (or cost) for such problems cannot be piecewise — a crucial limitation preventing the application of such solutions to many planning and operations research problems.

In this paper, we provide an exact *symbolic dynamic programming* (SDP) solution to a useful subset of continuous state and action Markov decision processes (CSA-MDPs) with *multivariate* continuous state and actions, discrete noise, *piecewise* linear dynamics, and *piecewise* linear (or restricted *piecewise* quadratic) reward. To be concrete about the form of CSA-MDPs we can solve with our SDP approach, let us formalize a simple MARS ROVER problem:[1]

Example (MARS ROVER)**.** *A Mars Rover state consists of its continuous position x along a given route. In a given time step, the rover may move a continuous distance $y \in [-10, 10]$. The rover receives its greatest reward for taking a picture at $x = 0$, which quadratically decreases to zero at the boundaries of the range $x \in [-2, 2]$. The rover will automatically take a picture when it starts a time step within the range $x \in [-2, 2]$ and it only receives this reward once.*

Using boolean variable $b \in \{0, 1\}$ to indicate if the picture has already been taken ($b = 1$), x' and b' to denote post-action state, and R to denote reward, we express the MARS ROVER CSA-MDP using piecewise dynamics and reward:

$$P(b'{=}1|x, b) = \begin{cases} b \vee (x \geq -2 \wedge x \leq 2) : & 1.0 \\ \neg b \wedge (x < -2 \vee x > 2) : & 0.0 \end{cases} \quad (1)$$

$$P(x'|x, y) = \delta \left(x' - \begin{cases} y \geq -10 \wedge y \leq 10 : & x + y \\ y < -10 \vee y > 10 : & x \end{cases} \right) \quad (2)$$

$$R(x, b) = \begin{cases} \neg b \wedge x \geq -2 \wedge x \leq 2 : & 4 - x^2 \\ b \vee x < -2 \vee x > 2 : & 0 \end{cases} \quad (3)$$

Then there are two natural questions that we want to ask:

(a) What is the optimal form of value that can be obtained from any state over a fixed time horizon?

(b) What is the corresponding closed-form optimal policy?

[1] For purposes of concise exposition and explanation of the optimal value function and policy, this CSA-MDP example uses continuous univariate state and action and deterministic transitions; the empirical results will later discuss a range of CSA-MDPs with multivariate continuous state and action and stochastic transitions.

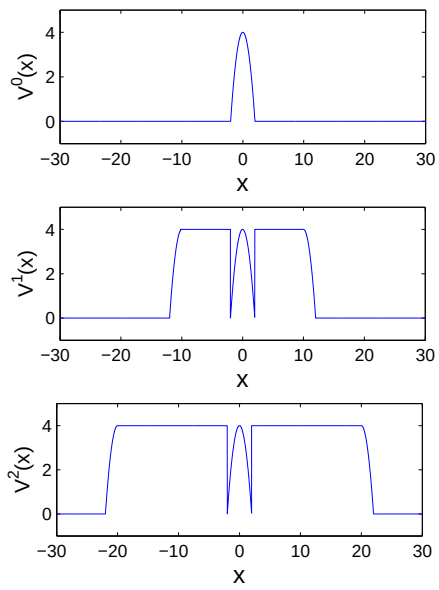

Figure 1: Optimal sum of rewards (value) $V^t(x)$ for $b = 0$ (*false*) for time horizons (i.e., decision stages remaining) $t = 0$, $t = 1$, and $t = 2$ on the MARS ROVER problem. For $x \in [-2, 2]$, the rover automatically takes a picture and receives a reward quadratic in x. We initialized $V^0(x, b) = R(x, b)$; for $V^1(x)$, the rover achieves non-zero value up to $x = \pm 12$ and for $V^2(x)$, up to $x = \pm 22$.

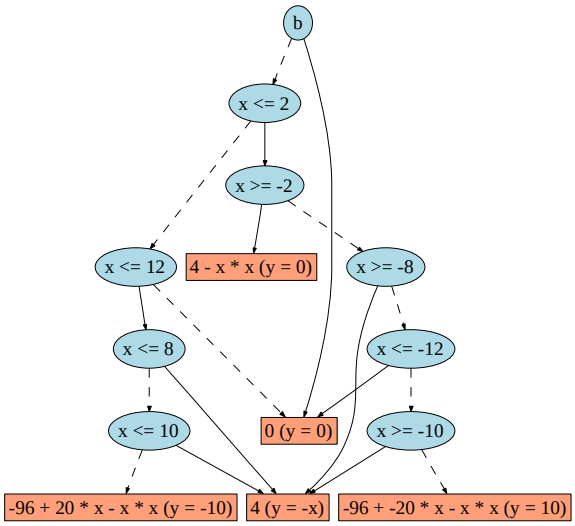

Figure 2: Optimal value function $V^1(x)$ for the MARS ROVER problem represented as an extended algebraic decision diagram (XADD). Here the solid lines represent the *true* branch for the decision and the dashed lines the *false* branch. To evaluate $V^1(x)$ for any state x, one simply traverses the diagram in a decision-tree like fashion until a leaf is reached where the non-parenthetical expression provides the *optimal value* and the parenthetical expression provides the *optimal policy* ($y = \pi^{*,1}(x)$) to achieve value $V^1(x)$.

To get a sense of the form of the optimal solution to problems such as MARS ROVER, we present the 0-, 1-, and 2-step time horizon solutions for this problem in Figure 1; further, in symbolic form, we display both the 1-step time horizon value function (the 2-step is too large to display) *and* corresponding optimal policy in Figure 2. Here, the piecewise nature of the transition and reward function leads to piecewise structure in the value function and policy. Yet despite the intuitive and simple nature of this result, we are unaware of prior methods that can produce such exact solutions.

To this end, we extend the previous SDP framework of (Sanner, Delgado, and de Barros 2011) to the case of continuous actions to obtain the *optimal closed-form* value function and policy for the class of CSA-MDPs described previously (as well as the useful deterministic subset). As the fundamental technical contribution of the paper, we show how the continuous action maximization step in the dynamic programming backup can be evaluated optimally and symbolically — a task which amounts to *symbolic* constrained optimization subject to unknown state parameters; we further integrate this technique to work with an efficient and compact data structure for SDP — the extended algebraic decision diagram (XADD). In addition to the solution of the nonlinear MARS ROVER planning example above, we demonstrate empirical results on RESERVOIR MANAGEMENT and INVENTORY CONTROL domains from operations research to show the *first automated exact solution* to these problems.

Continuous State and Action MDPs

In our CSA-MDPs, states are represented by vectors of variables $(\vec{b}, \vec{x}) = (b_1, \ldots, b_n, x_1, \ldots, x_m)$. We assume that each $b_i \in \{0, 1\}$ $(1 \le i \le n)$ is boolean and each $x_j \in \mathbb{R}$ $(1 \le j \le m)$ is continuous. We also assume a finite set of p actions $A = \{a_1(\vec{y}_1), \ldots, a_p(\vec{y}_p)\}$, where $\vec{y}_k \in \mathbb{R}^{|\vec{y}_k|}$ $(1 \le k \le p)$ denote continuous parameters for action a_k.

A CSA-MDP model requires the following: (i) a joint state transition model $P(\vec{b}', \vec{x}' | \cdots, a, \vec{y})$, which specifies the probability of the next state $(\vec{b}', \vec{x}')$ conditioned on a subset of the previous and next state and action $a(\vec{y})$; (ii) a reward function $R(\vec{b}, \vec{x}, a, \vec{y})$, which specifies the immediate reward obtained by taking action $a(\vec{y})$ in state $(\vec{b}, \vec{x})$; and (iii) a discount factor γ, $0 \le \gamma \le 1$. A policy π specifies the action $a(\vec{y}) = \pi(\vec{b}, \vec{x})$ to take in each state $(\vec{b}, \vec{x})$. Our goal is to find an optimal sequence of finite horizon-dependent policies $\Pi^* = (\pi^{*,1}, \ldots, \pi^{*,H})$ that maximizes the expected sum of discounted rewards over a horizon $h \in H; H \ge 0$:

$$V^{\Pi^*}(\vec{x}) = E_{\Pi^*}\left[\sum_{h=0}^{H} \gamma^h \cdot r^h \middle| \vec{b}_0, \vec{x}_0\right]. \quad (4)$$

Here r^h is the reward obtained at horizon h following Π^* where we assume starting state $(\vec{b}_0, \vec{x}_0)$ at $h = 0$.

CSA-MDPs as defined above are naturally factored (Boutilier, Dean, and Hanks 1999) in terms of state variables $(\vec{b}, \vec{x}, \vec{y})$; as such, transition structure can be exploited in the form of a dynamic Bayes net (DBN) (Dean and Kanazawa 1989) where the conditional probabilities $P(b_i' | \cdots)$ and $P(x_j' | \cdots)$ for each next state variable can condition on the action, current and next state. We assume there are no *synchronic arcs* (variables that condition on each other in the same time slice) within the binary $\vec{b}$ or continuous variables $\vec{x}$, but we allow synchronic arcs from

$\vec{b}$ to $\vec{x}$.[2] Hence we can factorize the joint transition model as

$$P(\vec{b}', \vec{x}' | \vec{b}, \vec{x}, a, \vec{y}) = \prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a, \vec{y}) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y}).$$

We call the conditional probabilities $P(b_i' | \vec{b}, \vec{x}, a, \vec{y})$ for *binary* variables b_i ($1 \leq i \leq n$) conditional probability functions (CPFs) — not tabular enumerations — because in general these functions can condition on both discrete and continuous state as in (1). For the *continuous* variables x_j ($1 \leq j \leq m$), we represent the CPFs $P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y})$ with *piecewise linear equations* (PLEs) satisfying three properties: (i) PLEs can only condition on the action, current state, and previous state variables, (ii) PLEs are deterministic meaning that to be represented by probabilities they must be encoded using Dirac $\delta[\cdot]$ functions (example forthcoming), and (iii) PLEs are piecewise linear, where the piecewise conditions may be arbitrary logical combinations of $\vec{b}$, $\vec{b}'$ and linear inequalities over $\vec{x}$. An example PLE has been provided in (2) where the use of the $\delta[\cdot]$ function ensures that this is a conditional probability function that integrates to 1 over x'; In more intuitive terms, one can see that this $\delta[\cdot]$ is a simple way to encode the PLE transition $x' = \{\ldots$ in the form of $P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y})$.

While it will be clear that our restrictions do not permit general stochastic transition noise (e.g., Gaussian noise as in LQG control), they do permit discrete noise in the sense that $P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y})$ may condition on $\vec{b}'$, which are stochastically sampled according to their CPFs. We note that this representation effectively allows modeling of continuous variable transitions as a mixture of δ functions, which has been used frequently in previous exact continuous state MDP solutions (Feng et al. 2004; Meuleau et al. 2009).

We allow the reward function $R(\vec{b}, \vec{x}, a, \vec{y})$ to be either (i) a general piecewise linear function (boolean or linear conditions and linear values) such as

$$R(\vec{b}, \vec{x}, a, \vec{y}) = \begin{cases} b \wedge x_1 \leq x_2 + 1 : & 1 - x_1 + 2x_2 \\ \neg b \vee x_1 > x_2 + 1 : & 3x_1 + 2x_2 \end{cases} \quad (5)$$

or (ii) a piecewise quadratic function of univariate state and a linear function of univariate action parameters as demonstrated in MARS ROVER (3). These transition and reward constraints will ensure that all derived functions in the solution of these CSA-MDPs adhere to the reward constraints.

Solution Methods

Now we provide a continuous state generalization of *value iteration* (Bellman 1957), which is a dynamic programming algorithm for constructing optimal policies. It proceeds by constructing a series of h-stage-to-go value functions $V^h(\vec{b}, \vec{x})$. Initializing $V^0(\vec{b}, \vec{x}) = 0$) we define the quality $Q_a^h(\vec{b}, \vec{x}, \vec{y})$ of taking action $a(\vec{y})$ in state $(\vec{b}, \vec{x})$ and acting so as to obtain $V^{h-1}(\vec{b}, \vec{x})$ thereafter as the following:

$$Q_a^h(\vec{b}, \vec{x}, \vec{y}) = \Bigg[R(\vec{b}, \vec{x}, a, \vec{y}) + \gamma \cdot \sum_{\vec{b}'} \int \left(\prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a, \vec{y}) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y}) \right) V^{h-1}(\vec{b}', \vec{x}') d\vec{x}' \Bigg] \quad (6)$$

Given $Q_a^h(\vec{b}, \vec{x})$ for each $a \in A$, we can proceed to define the h-stage-to-go value function as follows:

$$V^h(\vec{b}, \vec{x}) = \max_{a \in A} \max_{\vec{y} \in \mathbb{R}^{|\vec{y}|}} \left\{ Q_a^h(\vec{b}, \vec{x}, \vec{y}) \right\} \quad (7)$$

If the horizon H is finite, then the optimal value function is obtained by computing $V^H(\vec{b}, \vec{x})$ and the optimal horizon-dependent policy $\pi^{*,h}$ at each stage h can be easily determined via $\pi^{*,h}(\vec{b}, \vec{x}) = \arg\max_a \arg\max_{\vec{y}} Q_a^h(\vec{b}, \vec{x}, \vec{y})$. If the horizon $H = \infty$ and the optimal policy has finitely bounded value, then value iteration can terminate at horizon h if $V^h = V^{h-1}$; then $V^\infty = V^h$ and $\pi^{*,\infty} = \pi^{*,h}$.

From this *mathematical* definition, we next show how to *compute* (6) and (7) for the previously defined CSA-MDPs.

Symbolic Dynamic Programming (SDP)

In this section, we extend the *symbolic dynamic programming* (SDP) work of (Sanner, Delgado, and de Barros 2011) to the case of continuously parameterized actions for CSA-MDPs. We present the general SDP framework for value iteration in Algorithm 1 (`VI`) and a regression subroutine in Algorithm 2 (`Regress`) where we have omitted parameters $\vec{b}$ and $\vec{x}$ from V and Q to avoid notational clutter. The difference between this SDP algorithm and that in (Sanner, Delgado, and de Barros 2011) comes in the continuous action parameter maximization in line 7 of `VI`. But first we recap SDP, which uses the *case* representation and operations.

Case Representation and Operators

From here out, we assume that all symbolic functions can be represented in *case* form (Boutilier, Reiter, and Price 2001):

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \quad (8)$$

Here the ϕ_i are logical formulae defined over the state $(\vec{b}, \vec{x})$ that can include arbitrary logical $(\wedge, \vee, \neg)$ combinations of (i) boolean variables and (ii) *linear* inequalities $(\geq, >, \leq, <)$ over continuous variables. Each ϕ_i will be disjoint from the other ϕ_j ($j \neq i$); however the ϕ_i may not exhaustively cover the state space, hence f may only be a *partial function* and may be undefined for some variable assignments. The f_i may be either linear or quadratic in the continuous parameters according to the same restrictions as for $R(\vec{b}, \vec{x}, a, \vec{y})$. We require f to be continuous (including no discontinuities at partition boundaries); operations preserve this property.

Unary operations such as scalar multiplication $c \cdot f$ (for some constant $c \in \mathbb{R}$) or negation $-f$ on case statements f

[2] Synchronic arcs between variables within $\vec{b}$ or within $\vec{x}$ can be accommodated if the forthcoming Algorithm 2 (`Regress`) is modified to multiply and marginalize-out multiple next-state variables in one elimination step according to the DBN structure.

Algorithm 1: VI(CSA-MDP, H) $\longrightarrow (V^h, \pi^{*,h})$

1 **begin**
2 $\quad V^0 := 0, h := 0$
3 $\quad$ **while** $h < H$ **do**
4 $\quad\quad h := h + 1$
5 $\quad\quad$ **foreach** $a(\vec{y}) \in A$ **do**
6 $\quad\quad\quad Q_a^h(\vec{y})$:= Regress $(V^{h-1}, a, \vec{y})$
7 $\quad\quad\quad Q_a^h := \max_{\vec{y}} Q_a^h(\vec{y})$ *// Continuous* max
8 $\quad\quad\quad V^h := \text{casemax}_a\, Q_a^h$ *// casemax all* Q_a
9 $\quad\quad\quad \pi^{*,h} := \arg\max_{(a,\vec{y})} Q_a^h(\vec{y})$
10 $\quad\quad$ **if** $V^h = V^{h-1}$ **then**
11 $\quad\quad\quad$ break *// Terminate if early convergence*
12 $\quad\quad$
13 $\quad$ **return** $(V^h, \pi^{*,h})$
14 **end**

Algorithm 2: Regress$(V, a, \vec{y}) \longrightarrow Q$

1 **begin**
2 $\quad Q =$ Prime (V) *// All* $b_i \rightarrow b_i'$ *and all* $x_i \rightarrow x_i'$
3 $\quad$ *// Continuous regression marginal integration*
4 $\quad$ **for** *all* x_j' *in* Q **do**
5 $\quad\quad Q := \int Q \otimes P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y})\, d_{x_j'}$
6 $\quad$ *// Discrete regression marginal summation*
7 $\quad$ **for** *all* b_i' *in* Q **do**
8 $\quad\quad Q := \left[Q \otimes P(b_i' | \vec{b}, \vec{x}, a, \vec{y}) \right] |_{b_i'=1}$
9 $\quad\quad\quad \oplus \left[Q \otimes P(b_i' | \vec{b}, \vec{x}, a, \vec{y}) \right] |_{b_i'=0}$
10 $\quad$ **return** $R(\vec{b}, \vec{x}, a, \vec{y}) \oplus (\gamma \otimes Q)$
11 **end**

are simply applied to each f_i $(1 \leq i \leq k)$. Intuitively, to perform a *binary operation* on two case statements, we simply take the cross-product of the logical partitions of each case statement and perform the corresponding operation on the resulting paired partitions. Letting each ϕ_i and ψ_j denote generic first-order formulae, we can perform the "cross-sum" $\oplus$ of two (unnamed) cases in the following manner:

$$\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} \oplus \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{cases}$$

Likewise, we perform $\ominus$ and $\otimes$ by, respectively, subtracting or multiplying partition values (as opposed to adding them) to obtain the result. Some partitions resulting from case operators may be inconsistent (infeasible) and removed.

Next we define *symbolic case maximization*:

$$\text{casemax}\left(\begin{cases} \phi_1 : f_1 \\ \phi_2 : f_2 \end{cases}, \begin{cases} \psi_1 : g_1 \\ \psi_2 : g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge f_1 > g_1 : f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \leq g_1 : g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \leq g_2 : g_2 \\ \vdots \qquad\qquad\qquad\quad \vdots \end{cases}$$

If all f_i and g_i are linear, the casemax result is clearly still linear. If the f_i or g_i are quadratic according to the previous reward restriction, it will shortly become obvious that the expressions $f_i > g_i$ or $f_i \leq g_i$ will be at most univariate quadratic and any such constraint can be *linearized* into a combination of at most two linear inequalities (unless tautologous or inconsistent) by completing the square (e.g., $-x^2 + 20x - 96 > 0 \equiv [x - 10]^2 \leq 4 \equiv [x > 8] \wedge [x \leq 12]$). Hence according to the earlier restrictions, the result of this casemax operator will be representable in the case format previously described (i.e., linear inequalities in decisions).

There are two operations in Regress that we have not defined yet. The first operation of *boolean restriction* required in lines 8–9 is obvious and an example is omitted: in this operation $f|_{b=v}$, anywhere a boolean variable b occurs in f, we assign it the value $v \in \{0, 1\}$. The second operation of *continuous integration* $\int Q(x_j') \otimes P(x_j' | \cdots) dx_j'$ is required in line 5; as previously defined, $P(x_j' | \cdots)$ will always be of the form $\delta[x_j' - h(\vec{z})]$ where $h(\vec{z})$ is a case statement and $\vec{z}$ does not contain x_j'. Rules of integration then tell us that $\int f(x_j') \otimes \delta[x_j' - h(\vec{z})] dx_j' = f(x_j')\{x_j'/h(\vec{z})\}$ where the latter operation indicates that any occurrence of x_j' in $f(x_j')$ is *symbolically substituted* with the case statement $h(\vec{z})$. The full specification of this operation was a key contribution of (Sanner, Delgado, and de Barros 2011) so we refer the reader to that paper for further technical details. The important insight is that this $\int$ operation yields a result that is a case statement in the form previously outlined.

Maximization of Continuous Action Parameters

The only operation in VI and Regress that has not yet been defined is the continuous action maximization in line 7 of VI that forms the key novel contribution of this paper. Reintroducing suppressed state variables and renaming Q_a^h to f, we write this operation as $g(\vec{b}, \vec{x}) := \max_{\vec{y}} f(\vec{b}, \vec{x}, \vec{y})$ — crucially we note that *the* maximizing $\vec{y}$ is a function $g(\vec{b}, \vec{x})$, hence requiring *symbolic* constrained optimization.

From here out we assume that all case partition conditions ϕ_i of f consist of conjunctions of non-negated linear inequalities and possibly negated boolean variables — conditions easy to enforce since negation inverts inequalities, e.g., $\neg[x < 2] \equiv [x \geq 2]$ and disjunctions can be split across multiple non-disjunctive, disjoint case partitions.

Exploiting the commutativity of max, we can first rewrite any multivariate $\max_{\vec{y}}$ as a sequence of univariate max operations $\max_{y_1} \cdots \max_{y_{|\vec{y}|}}$; hence it suffices to provide just the *univariate* $\max_y$ solution: $g(\vec{b}, \vec{x}) := \max_y f(\vec{b}, \vec{x}, y)$.

We can rewrite $f(\vec{b}, \vec{x}, y)$ via the following equalities:

$$\begin{aligned} \max_y f(\vec{b}, \vec{x}, y) &= \max_y \text{casemax}_i\, \phi_i(\vec{b}, \vec{x}, y) f_i(\vec{b}, \vec{x}, y) \\ &= \text{casemax}_i \boxed{\max_y \phi_i(\vec{b}, \vec{x}, y) f_i(\vec{b}, \vec{x}, y)} \end{aligned} \quad (9)$$

The first equality is a consequence of the mutual disjointness of the partitions in f. Then because $\max_y$ and casemax_i are commutative and may be reordered, we can compute $\max_y$ for *each case partition individually*. Thus to

complete this section we need only show how to symbolically compute a single partition $\max_y \phi_i(\vec{b},\vec{x},y) f_i(\vec{b},\vec{x},y)$.

To make the partition maximization procedure concrete, we use an example that arises in the MARS ROVER problem. This partition i (resulting from applying SDP) has conditions $\phi_i(x,b,y) \equiv \neg b \wedge (x \geq 2) \wedge (y \leq 10) \wedge (y \geq -10) \wedge (y \leq 2-x) \wedge (y \geq -2-x)$ and value $f_i(x,y) = 4-(x+y)^2$.

In ϕ_i, we observe that each conjoined constraint serves one of three purposes: (i) *upper bound on* y: it can be written as $y < \cdots$ or $y \leq \cdots$ (i.e., $y \leq 10$, $y \leq 2-x$), (ii) *lower bound on* y: it can be written as $y > \cdots$ or $y \geq \cdots$ (i.e., $d \geq -10$, $d \geq x-2$)[3] or (iii) *independent of* y: the constraints do not contain y and can be safely factored outside of the $\max_y$ (i.e., $Ind = \neg b \wedge (x \geq 2)$). Because there are multiple symbolic upper and lower bounds on y, in general we will need to apply the casemax (casemin) operator to determine the highest lower bound LB (lowest upper bound UB):

$$LB = \text{casemax}(-10, -2-x) = \begin{cases} x \leq 8: & -2-x \\ x > 8: & -10 \end{cases}$$

$$UB = \text{casemin}(10, 2-x) = \begin{cases} x > -8: & 2-x \\ x \leq -8: & 10 \end{cases}$$

We know that $\max_y \phi_i(\vec{b},\vec{x},y) f_i(\vec{b},\vec{x},y)$ for a continuous function f_i (here at most quadratic) must occur at the critical points of the function — either the upper or lower bounds (UB and LB) of y, or the $Root$ (i.e., zero) of $\frac{\partial}{\partial y} f_i$ w.r.t. y (because f_i is at most quadratic, there exists at most one $Root$). Each of UB, LB, and $Root$ is a symbolic function of $\vec{b}$ and $\vec{x}$; here we show the computation of $Root$:

$$\frac{\partial}{\partial y} f_i = -2y - 2d = 0 \quad \Longrightarrow \quad Root = y = -x$$

Given the *potential* maxima points of $y = UB$, $y = LB$, and $y = Root$ of $\frac{\partial}{\partial y} f_i(\vec{b},\vec{x},y)$ w.r.t. constraints $\phi_i(\vec{b},\vec{x},y)$ — which are all symbolic functions — we must symbolically evaluate which yields the maximizing value Max for this case partition:

$$Max = \begin{cases} \exists Root: & \text{casemax}(f_i\{y/Root\}, f_i\{y/UB\}, f_i\{y/LB\}) \\ \text{else:} & \text{casemax}(f_i\{y/UB\}, f_i\{y/LB\}) \end{cases}$$

Here $\text{casemax}(f,g,h) = \text{casemax}(f, \text{casemax}(g,h))$. The substitution operator $\{y/f\}$ replaces y with case statement f, defined in (Sanner, Delgado, and de Barros 2011).

For our running example, space precludes showing the final Max, so we show the pre-casemax operands instead:

$$Max = \text{casemax}\Big(f_i\{y/Root\} = 4-(x+-x)^2 = 4 \ ,$$

$$f_i\{y/LB\} = \begin{cases} x \leq 8: 4-(x+[-2-x])^2 = 0 \\ x > 8: 4-(x+[-10])^2 = -x^2+20x-96 \end{cases} ,$$

$$f_i\{y/UB\} = \begin{cases} x > -8: 4-(x+[2-x])^2 = 0 \\ x \leq -8: 4-(x+[10])^2 = -x^2-20x-96 \end{cases} \Big)$$

[3]For purposes of evaluating a case function f at an upper or lower bound, it does not matter whether a bound is inclusive ($\leq$ or $\geq$) or exclusive ($<$ or $>$) since f is required to be continuous and hence evaluating at the limit of the inclusive bound will match the evaluation for the exclusive bound.

Substituted values are subject to conditions in the cases being substituted and shown above in $[\cdot]$. When the casemax is evaluated, the resulting case conditions will have quadratic constraints like $-x^2+20x-96 > 0$, which must be linearized as previously discussed and shown for this example.

At this point, we have almost completed the computation of the $\max_y \phi_i(\vec{b},\vec{x},y) f_i(\vec{b},\vec{x},y)$ except for one issue: the incorporation of the Ind constraints (factored out previously) and additional constraints that arise from the symbolic nature of the UB, LB, and $Root$. Specifically for the latter, we need to ensure that indeed $LB \leq Root \leq UB$ (or if no root exists, then $LB \leq UB$) by building a set of constraints $Cons$ that ensure these conditions hold; to do this, it suffices to ensure that for each possible expression e used to construct LB that $e \leq Root$ and similarly for the $Root$ and UB. For the running MARS ROVER example:

$$Cons = \underbrace{[-2-x \leq -x] \wedge [-10 \leq -x]}_{LB \leq Root} \wedge \underbrace{[-x \leq 2-x] \wedge [-x \leq 10]}_{Root \leq UB}$$

Here, two constraints are tautologies and may be removed.

Now we express the final result as a single case partition:

$$\max_y \phi_i(\vec{b},\vec{x},y) f_i(\vec{b},\vec{x},y) \quad = \quad \{ Cons \wedge Ind : Max$$

Returning to (9), we find that we have now specified the inner operation (shown in the $\square$). Hence, to complete the maximization for an entire case statement f, we need only apply the above procedure to each case partition of f and then casemax all of these results. Revisiting the MARS ROVER example V^1 in Figure 2, we can observe many of the decision inequalities and value expressions from the above example. To obtain the policy in Figure 2, we need only annotate leaf values with any UB, LB, and $Root$ substitutions.

Extended ADDs (XADDs) (Sanner, Delgado, and de Barros 2011) extension of ADDs (Bahar et al. 1993) provides a compact data structure to support case statements and operations. Using XADDs in SDP as a continuous version of the ADD-based SPUDD (Hoey et al. 1999) algorithm for discrete MDPs, we maintain compact forms of Q and V, e.g., as shown in V^2 for MARS ROVER in Figure 2. XADDs also permit the use of linear constraint feasibility checkers from LP solvers to prune unreachable paths in the XADD.

The only operation that has not been previously defined for XADDs is $\max_y$, but this is easy: treating each XADD path from root to leaf node as a single case partition with conjunctive constraints, $\max_y$ is performed at each leaf subject to these constraints and all path $\max_y$'s are then accumulated via the casemax operation to obtain the final result.

Empirical Results

We evaluated SDP using XADDs on the didactic nonlinear MARS ROVER example and two problems from Operations Research (OR) — INVENTORY CONTROL and RESERVOIR MANAGEMENT— described below.[4] Space precludes showing more results for MARS ROVER than in Figures 1 and 2; we note that SDP efficiently solves it for arbitrary horizons.

[4]Full problem specifications and Java code to reproduce these experiments are currently in Google Code; anonymity restrictions prevent us from providing this link in the submitted paper version.

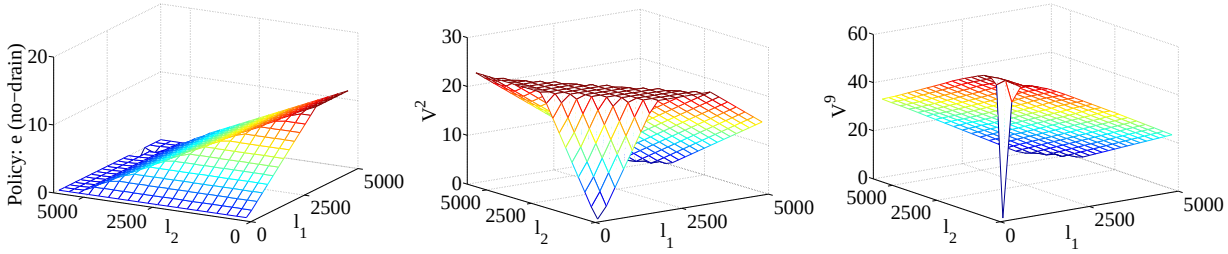

Figure 3: RESERVOIR MANAGEMENT: *(left)* Policy $no\text{-}drain(e) = \pi^{2,*}(l_1, l_2)$ for elapsed time e; *(middle)* $V^2(l_1, l_2)$; *(right)* $V^9(l_1, l_2)$.

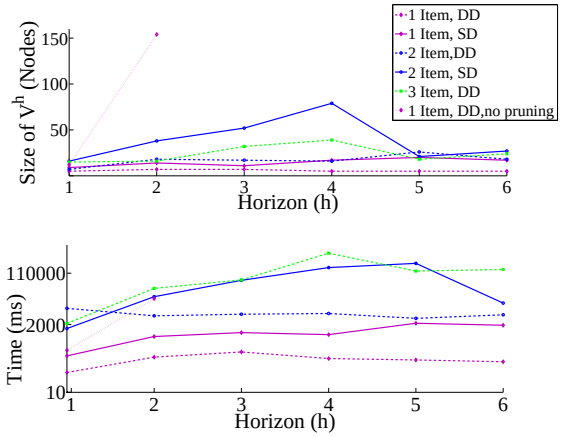

Figure 4: INVENTORY CONTROL: space and time vs. horizon.

INVENTORY CONTROL: Inventory control problems (how much of an item to reorder subject to capacity constraints, demand, and optimization criteria) date back to the 1950's with Scarf's seminal optimal solution to the *single-item capacitated inventory control* (SCIC) problem (Arrow, Karlin, and Scarf 1958). *Multi-item joint capacitated inventory* (MJCIC) control (upper limits on total storage of all items) has proved to be an NP-hard problem and as a consequence, most solutions resort to some form of approximation (Bitran and Yanasse 1982; Wu, Shi, and Duffie 2010); indeed, we are unaware of any work which claims to find an exact closed-form non-myopic optimal policy for *all* (continuous) inventory states for MJCIC under linear reordering costs and linear holding costs; these problems can be easily modeled as CSA-MDPs and solved optimally with SDP.

In Figure 4, we provide a time and space analysis of deterministic- and stochastic-demand (resp. DD and SD) variants of the SCIC and MJCIC problem for up to three items (the same scale of problems often studied in the OR literature); for each number of items $n \in \{1, 2, 3\}$ the state (inventory levels) is $\vec{x} \in [0, \infty]^n$ and the action (reorder amounts) is $\vec{y} \in [0, \infty]^n$. Orders are made at one month intervals and we solve for a horizon up to $h = 6$ months. Here we see that linear feasbility checking/pruning in the XADD is crucial – we cannot solve beyond $h = 2$ without it for 1 item! While solving for larger numbers of items and SD (rather than DD) both increase time and space, the solutions quickly reach quiescence indicating structural convergence.

RESERVOIR MANAGEMENT: Reservoir management is also well-studied in OR (Mahootchi 2009; Yeh 1985). The key continuous decision is how much elapsed time e to *drain* (or *not drain*) each reservoir to maximize electricity revenue over the decision-stage horizon while avoiding reservoir overflow and underflow. Cast as a CSA-MDP, we believe SDP provides the first approach capable of deriving an exact closed-form non-myopic optimal policy for all levels.

We examine a 2-reservoir problem with respective levels $(l_1, l_2) \in [0, \infty]^2$ with reward penalties for overflow and underflow and a reward gain linear in the elapsed time e for electricity generated in periods when the $drain(e)$ action drains water from l_2 to l_1 (the other action is $no\text{-}drain(e)$); we assume deterministic rainfall replenishment. In Figure 3, we show a plot of the optimal closed-form policy at $h = 2$: the solution interleaves $drain(e)$ and $no\text{-}drain(e)$ where even horizons are the latter; here we see that we avoid draining for the longest elapsed time e when l_2 is low (wait for rain to replenish) and l_1 is high (draining water into it could overflow it). $V^2(l_1, l_2)$ and $V^9(l_1, l_2)$ show the progression of convergence from horizon $h = 2$ to $h = 9$ — low levels of l_1 and l_2 allow the system to generate electricity for the longest total elapsed time over 9 decision stages.

Related Work and Concluding Remarks

This work is an extension of SDP (Sanner, Delgado, and de Barros 2011) that handled continuous state and *discrete actions*, which itself built on the continuous state, discrete action work of (Boyan and Littman 2001; Feng et al. 2004; Li and Littman 2005). In control theory, linear-quadratic Gaussian (LQG) control (Athans 1971) using linear dynamics with continuous actions, Gaussian noise, and quadratic reward is most closely related. However, these exact solutions do not extend to discrete and continuous systems with *piecewise* dynamics or reward as shown in (1), (2), and (3). Combining this work with initial state focused techniques (Meuleau et al. 2009) and focused approximations that exploit optimal value structure (St-Aubin, Hoey, and Boutilier 2000) or further afield (Remi Munos 2002; Kveton, Hauskrecht, and Guestrin 2006; Marecki, Koenig, and Tambe 2007) are promising directions for future work.

We have presented an SDP solution for continuous state *and* action MDPs with the key contribution of *symbolic constrained optimization* to solve the continuous action maximization problem. We believe this is the first work to propose optimal closed-form solutions to MDPs with *multivariate* continuous state *and* actions, discrete noise, *piecewise* linear dynamics, and *piecewise* linear (or restricted *piecewise* quadratic) reward; further, we believe our experimental results are the first exact solutions to these problems to provide a closed-form optimal policy for all (continuous) states.

References

Arrow, K.; Karlin, S.; and Scarf, H. 1958. *Studies in the mathematical theory of inventory and production*. Stanford University Press.

Athans, M. 1971. The role and use of the stochastic linear-quadratic-gaussian problem in control system design. *IEEE Transaction on Automatic Control* 16(6):529–552.

Bahar, R. I.; Frohm, E.; Gaona, C.; Hachtel, G.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic Decision Diagrams and their applications. In *IEEE /ACM International Conference on CAD*.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bitran, G. R., and Yanasse, H. 1982. Computational complexity of the capacitated lot size problem. *Management Science* 28(10):1271–81.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR* 11:1–94.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *IJCAI-01*, 690–697.

Boyan, J., and Littman, M. 2001. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems NIPS-00*, 1026–1032.

Bresina, J. L.; Dearden, R.; Meuleau, N.; Ramkrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty:a challenge for ai. In *Uncertainty in Artificial Intelligence (UAI-02)*.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous markov decision problems. In *Uncertainty in Artificial Intelligence (UAI-04)*, 154–161.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *UAI-99*, 279–288.

Kveton, B.; Hauskrecht, M.; and Guestrin, C. 2006. Solving factored mdps with hybrid state and action variables. *Journal Artificial Intelligence Research (JAIR)* 27:153–201.

Lamond, B., and Boukhtouta, A. 2002. Water reservoir applications of markov decision processes. In *International Series in Operations Research and Management Science, Springer*.

Li, L., and Littman, M. L. 2005. Lazy approximation for solving continuous finite-horizon mdps. In *National Conference on Artificial Intelligence AAAI-05*, 1175–1180.

Mahootchi, M. 2009. *Storage System Management Using Reinforcement Learning Techniques and Nonlinear Models*. Ph.D. Dissertation, University of Waterloo,Canada.

Marecki, J.; Koenig, S.; and Tambe, M. 2007. A fast analytical algorithm for solving markov decision processes with real-valued resources. In *International Conference on Uncertainty in Artificial Intelligence IJCAI*, 2536–2541.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam. 2009. A heuristic search approach to planning with continuous resources in stochastic domains. *Journal Artificial Intelligence Research (JAIR)* 34:27–59.

Remi Munos, A. M. 2002. Variable resolution discretization in optimal control. *Machine Learning* 49, 2–3:291–323.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2011. Symbolic dynamic programming for discrete and continuous state mdps. In *Proceedings of the 27th Conference on Uncertainty in AI (UAI-2011)*.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2000. APRICODD: Approximate policy construction using decision diagrams. In *NIPS-2000*, 1089–1095.

Wu, T.; Shi, L.; and Duffie, N. A. 2010. An hnp-mp approach for the capacitated multi-item lot sizing problem with setup times. *IEEE T. Automation Science and Engineering* 7(3):500–511.

Yeh, W. G. 1985. Reservoir management and operations models: A state-of-the-art review. *Water Resources research* 21,12:17971818.