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 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 dyanmic 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:

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

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

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

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 define a range of CSA-MDPs with multivariate continuous state and action and stochastic transitions.

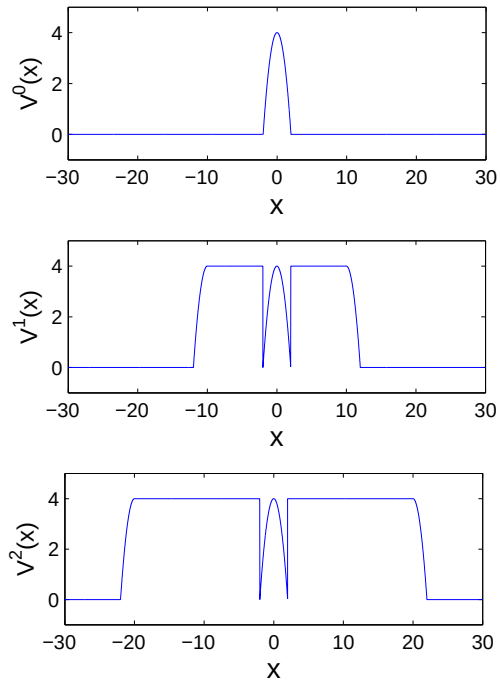

Figure 1: Optimal sum of rewards (value) $V^t(x)$ for $b = 0$ $(false)$ for time horizons (i.e., decision stages remaining) $t = 0$, $t = 1$, and $t = 2$ on the MARS ROVER problem. For $x \in [-2, 2]$, the rover automatically takes a picture and receives a reward quadratic in x. We initialized $V^0(x,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

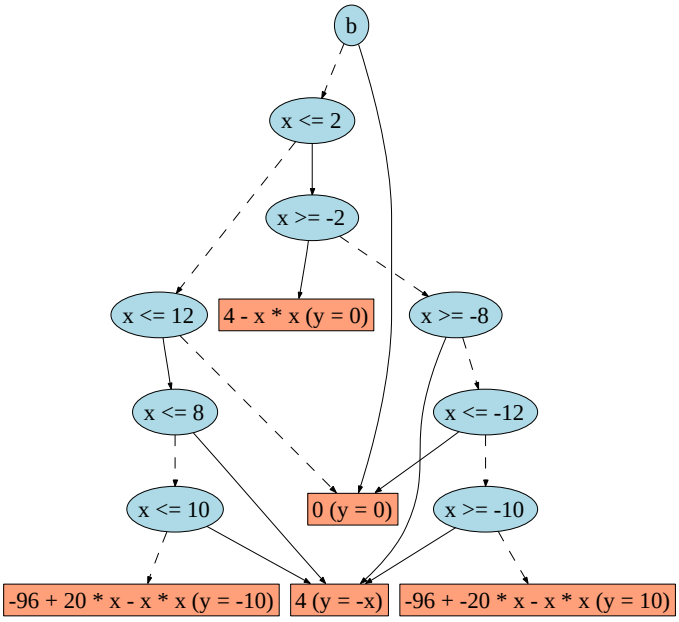

Figure 2: Optimal value function $V^1(x)$ for the MARS ROVER problem represented as an extended algebraic decision diagram (XADD). Here the solid lines represent the *true* branch for the decision and the dashed lines the *false* branch. To evaluate $V^1(x)$ for any state x, one simply traverses the 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 lead 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

Factored Representation

In our CSA-MDPs, states will be 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 $h \in H$; $H \ge 0$:

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

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 are first-order *Markov*, meaning that they can only condition on the action, current, 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 $\vec{b}$ 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). In the concluding remarks, we discuss the computational implications of relaxing the above restrictions on the transition and reward functions.

[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.

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:

$$Q_a^h(\vec{b}, \vec{x}, \vec{y}) = \Bigg[R(\vec{b}, \vec{x}, a, \vec{y}) + \gamma \cdot \quad (6)$$

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

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 Q-function regression subroutine 2 (`Regress`) where we have omitted parameters $\vec{b}$ and $\vec{x}$ from V and Q to avoid notational clutter. We note the single difference between this algorithm and that described in (Sanner, Delgado, and de Barros 2011) comes in the continuous action parameter maximization in line 7 of `VI`. Before we explain this contribution though, we first recap the SDP solution, which requires that the CSA-MDP and all other functions such as Q and V are represented in *case* form and all operations are *case operations*, defined next.

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 $(\geq, >, \leq, <)$

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

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

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

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

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 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 can perform $\ominus$ and $\otimes$ by, respectively, subtracting or multiplying partition values (as opposed to adding them) to obtain the result. Some partitions resulting from the application of the $\oplus$, $\ominus$, and $\otimes$ operators may be inconsistent (infeasible); we may simply discard such partitions as they are irrelevant to the function value.

For SDP, we'll also need to perform maximization, restriction, and substitution on case statements. *Symbolic case maximization* is fairly straightforward to define:

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

We remark that 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 quadratic inequality can be *linearized* into a logical combination of at most two linear inequalities 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 regression* $\int Q(x_j') \otimes P(x_j' | \cdots) dx_j'$ 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 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 $g(\vec{b}, \vec{x}) := \max_{\vec{y}} f(\vec{b}, \vec{x}, \vec{y})$ — the key point to realize here is that *the* maximizing $\vec{y}$ is a function $g(\vec{b}, \vec{x})$. We work through the derivation of g in this section, which amounts to *symbolic* constrained optimization subject to unknown state parameters $\vec{b}$ and $\vec{x}$.

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, e.g.,

$$f = \begin{cases} a \vee b : & f_1 \\ \neg a \wedge \neg b : & f_2 \end{cases} = \begin{cases} a : & f_1 \\ \neg a \wedge b : & f_1 \\ \neg a \wedge \neg b : & f_2 \end{cases} .$$

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:

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

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 $Root$). Here each of UB, LB, and $Root$ is a symbolic function of $\vec{b}$ and $\vec{x}$. UB and LB have already been computed, here we show the computation of $Root$:

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

[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 the exclusive and inclusive (limit) evaluations will match.

Finally, we have the potential maxima $y = UB$, $y = LB$, and $y = Root$ of $f_i(\vec{b}, \vec{x}, y)$ w.r.t. constraints $\phi_i(\vec{b}, \vec{x}, y)$, but since each of UB, LB, and $Root$ is a function of unassigned variables $\vec{b}$ and $\vec{x}$, we cannot substitute and numerically evaluate them to determine which is the true maximum. Instead, we symbolically evaluate this maximum Max as follows:

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

Here $\text{casemax}(f, g, h) = \text{casemax}(f, \text{casemax}(g, h))$ and 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:

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

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

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

When the casemax is executed, the quadratic values will enter the constraint inequalities and then require linearization as previously discussed; indeed, the linearization of $-x^2 + 20x - 96 > 0$ that will be required here has been provided previously.

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 $LB \leq Root \leq UB$ (of 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:

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

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 the original (9), we find that we have now specified the inner operation $\square$. Hence, to complete the maximization for an entire case statement f, we need only apply the above procedure to each case partition and then casemax together all of the results. And for the MARS ROVER example V^1 in Figure 2, one can gain an intuition as to where each of the decision equalities and value expressions has arisen from. On a final note, to obtain the policy, we simply annotate the case partition values with the substitutions that led to them during the above operations.

This concludes our fundamental contribution of the paper which computes the continuous action parameter maximization *symbolically* as a function of the state variables $\vec{b}$ and $\vec{x}$.

Extended ADDs (XADDs)

In practice, it can be prohibitively expensive to maintain a case statement representation of a value function with explicit partitions. Motivated by the SPUDD (Hoey et al. 1999) algorithm which maintains compact value function representations for finite discrete factored MDPs using algebraic decision diagrams (ADDs) (Bahar et al. 1993), Sanner *et al* (Sanner, Delgado, and de Barros 2011) introduced *extended ADDs* (XADDs) to handle continuous variables in an ADD-like data structure. An example XADD for the optimal MARS ROVERvalue function has already been provided in Figure 2 — partition constraints are simply represented as internal node decisions and partition values are simply represented as leaf nodes.

It is fairly straightforward for XADDs to support all case operations required for SDP. Standard operations like unary multiplication, negation, $\oplus$, and $\otimes$ are implemented exactly as they are for ADDs. The fact that the decision nodes have internal structure means that certain paths in the XADD may be inconsistent or infeasible (due to parent decisions). To remedy this, when the XADD has only linear decision nodes (as it will in this work), we can use a feasibility checker of a linear programming solver to prune out unreachable paths (Sanner, Delgado, and de Barros 2011), which substantially improves scalability as we show later.

Sanner *et al* (Sanner, Delgado, and de Barros 2011) outlined a number of issues that arise in XADDs along with efficient solutions; we refer the reader to that work for more details. In this work, we need only extend the XADD to support the previously defined continuous variable maximization operation — indeed, considering any path from root to leaf in an XADD as a case partition and value, the mapping of this operation from cases to XADDs is trivial and directly exploits the compact partition structure of the XADD.

Empirical Results

We implemented the CSA-MDP algorithm using the XADD representation and tested it on several domains. Apart from the MARS ROVER NONLINEAR, we considered problems from the OR literature such as INVENTORY CONTROLand RESERVOIR MANAGEMENT. In the following section we study these examples empirically.

INVENTORY CONTROL

This domain problem is a well-known optimization benchmark in the OR literature. The inventory control theory is concerned with the decision problems of when to buy and how much to buy of a certain inventory item (inventory quantity or lot-size). The goal is to optimize a certain property of the system such as costs or profits. (**?**)

There are various costs related to any item in the inventory. Production/ procurement costs are used to benefit ordering large quantities in advance. Holding/storage costs are used as the trade-off against production costs as holding items in the inventory before the time required is expensive. Shortage/ penalty costs are considered when the inventory can not meet the customer requirements. Demand levels are another source for holding inventories in case future requirements change deterministically or in a stochastic manner. We consider our model as a variation of Scarf's general inventory control model:

Scarf's (S,s) policy: The (S,s) policy is a wide spread policy used in practise which considers a policy with the two constant s,S such that:

$$y = \begin{cases} S & if x < s \\ x & if x \geq s \end{cases}$$

where y is the inventory level and x is the order quantity. A deterministic model of the inventory theory is defined so that all cost prices and demands are known with certainty.

The formal MDP definition of this problem is as below:

- y = State of the single-item inventory stock
- x = Action for ordering (quantity to order)
- Transition function $y = y + x - \varphi(t)$ where φ is the density of the demand distribution.
- Reward function $R = -C(x) = c(y - x) + L(y)$ which is the negative expected cost function with c(z) as the ordering cost and L(y) as the expected holding and shortage costs defined below:

$$L(y) = \int_0^y h(y - \xi)\varphi(\xi)\,\mathrm{d}xi + \int_y^\infty p(\xi - y)\varphi(\xi)\,\mathrm{d}\xi$$

where p() is the shortage cost function, h() the holding cost function and φ the density of the demand distribution.

We present a simple formulation of this problem with a maximum capacity C for the inventory. This capacitated version is proved to be an NP-hard problem (**?**) and more complicated than its uncapacitated counterparts. Assuming customer orders not satisfied in this month are backlogged for the next month, allows inventory to take negative values.

We consider three capacitated problem instances, a one product inventory with one order action, a two product inventory that needs two different orders with stochastic demand and a three product, 3 orders with deterministic demand. We present the mathematical formulation of the two product-two order case, the other two domains are modelled similarly. We take two continuous state variable x_1, x_2 indicating the current inventory quantity, with the total inventory capacity of 200, and a stochastic boolean state variable for customer demand level d where $d = 0$ is low demand levels (50) and $d = 1$ is high demand levels (150) according to some probability. The continuous action variable is the order quantity a_1, a_2.The transition for one of the state variables is defined below:

$$x_1' = \begin{cases} d \wedge (x_1 + a_1 + x_2 - 150 \leq 200): & x_1 + a_1 - 150 \\ d \wedge (x_1 + a_1 + x_2 - 150 \geq 200): & x_1 - 150 \\ \neg d \wedge (x_1 + a_1 + x_2 - 50 \leq 200): & x_1 + a_1 - 50 \\ \neg d \wedge (x_1 + a_1 + x_2 - 50 \geq 200): & x_1 - 50 \end{cases}$$

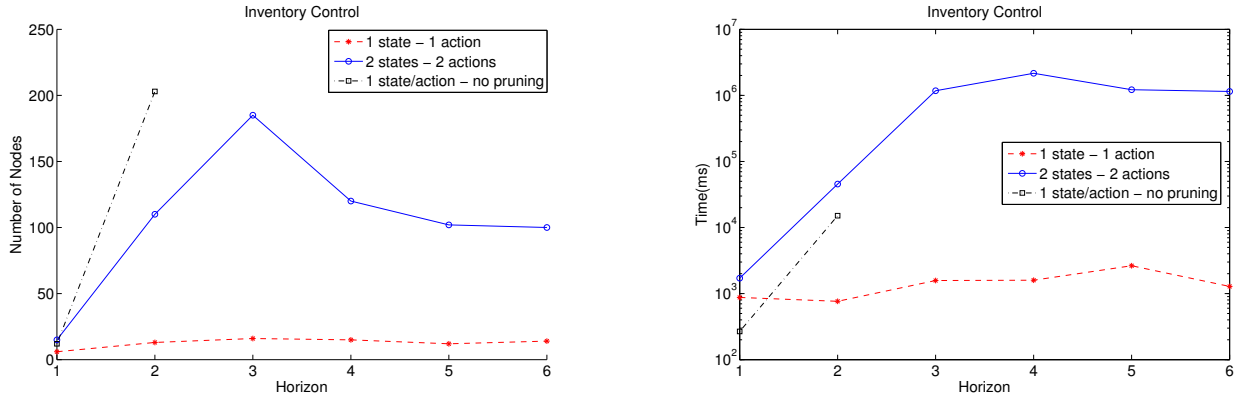

Figure 3: Space (# XADD nodes in value function) and time for different horizons of CSA-MDP on INVENTORY CONTROL Comparing 1D, 2D and no-pruning

The transition for the continuous actions partitions based on the maximum capacity of the inventory (for both products), and only adds the orders if the current total capacity (with respect to the orders of that product and the stocks available for both products) are less than this maximum capacity.The demand variable is transitioned stochastically.

$$d' = \begin{cases} d: & (0.7) \\ \neg d: & (0.3) \end{cases}$$

Next we define an immediate negative reward for the cost of producing an order and the storage cost of holding the products in the inventory and a positive reward for fulfilling the customer demand whenever there are enough stocks in the inventory. The total reward is the sum of the following two reward functions:

$$R_1 = \begin{cases} (x_1 \geq 150) \wedge d: & 150 - 0.5 \times a_1 - 0.1 \times x_1 \\ \neg(x_1 \geq 150) \wedge d: & 150 - 0.5 \times a_1 - 1.1 \times x_1 \\ (x_1 \geq 50) \wedge \neg d: & 50 - 0.5 \times a_1 - 0.1 \times x_1 \\ \neg(x_1 \geq 150) \wedge \neg d: & 50 - 0.5 \times a_1 - 1.1 \times x_1 \end{cases}$$

+

$$R_2 = \begin{cases} (x_2 \geq 150) \wedge d: & 150 - 0.4 \times a_2 - 0.1 \times x_2 \\ \neg(x_2 \geq 150) \wedge d: & 150 - 0.4 \times a_2 - 0.1 \times x_2 \\ (x_2 \geq 50) \wedge \neg d: & 50 - 0.4 \times a_2 - 0.1 \times x_2 \\ \neg(x_2 \geq 150) \wedge \neg d: & 50 - 0.4 \times a_2 - 0.1 \times x_2 \end{cases}$$

The reward function is based on demand levels and current stock in inventory. If the current stock is larger than the total inventory, we get rewarded for fulfilling the demand $(e.g 150)$, if the demand is high or if the inventory is not high enough, then the reward is $(e.g(150 - (x_1 + x_2)))$, in both cases action order costs are added. This allows the inventory to stock as many products as possible while not exceeding the capacity of the inventory.

We plot the results of comparing different INVENTORY CONTROLproblem sizes: 1-State,1-Action with stochastic demand (1SA,SD), 2-State,2-Action with stochastic and deterministic demand (2SA,SD ; 2SA,DD) and a 3-State, 3-Action with deterministic demand (3SA,DD). Figure 3 compares the time and space for different iterations for these problems instances and an extra comparison for the effect of not pruning on the 1D instance.

Comparing different problem sizes demonstrates the effect of the number of action variables in our algorithm. As the problem size increases, both action and states effect the time and space required to perform the algorithm. CSA-MDP claims exact results for these problem instances therefore it iterates on all possible state-action partitions to find the optimal policy.

The time and space have increased from the first iteration up to the third iteration for the 2D problem size, but then dropped for the next horizons due to pruning the XADD in our algorithm. As more constraints got added in for horizon 4, they cancelled the effects of some of the previous branches because of infeasibility and the pruning operation allows the XADD to grow smaller in space and requiring almost a constant time depending on the constraints added in each horizon.

For the 2-state-action problem, we considered two variations with constraints such as holding costs or total inventory capacity. Results show that adding holding costs to the problem does effect the complexity of the problem. In general adding state or action variables in the reward function can increase the XADD size. The holding cost adds the following constraint to all branches in the reward function: $-0.1 * x1 - 0.1 * x2$ In another simulation, we tested adding all the products to the inventory constraints such that the transition function is conditioned on both products and their future orders: $x_1' = (x_1 + a_1 + x_2 + a_2 - 150 \leq 200)$. This version requires more space since replacing each of the action variables with their related bounds adds to the decision nodes. Stochastic demands force more partitions for a problem instance as the plots for the 2-state-action case shows. Although the result for 1d stochastic is impressive, but for the 2d case, stochastic demand forces complexity. This partitioning occurs in the Q-function where the final result con-

siders the restriction to the stochastic variable values.

Without considering pruning, even the 1 product problem instance quickly falls into the curse of dimensionality problem. In fact, after the second iteration time and space grows exponentially and for this reason the plot fails to show the next time and space.

RESERVOIR MANAGEMENT

In this experiment, we consider a continuous-time approach for the multi-reservoir domain. The problem of RESERVOIR MANAGEMENTneeds to make an optimal decision on how much and when to discharge water from water reservoirs to maximize hydroelectric energy productions while considering environment constraints such as irrigation requirements and flood prevention.

A multi-reservoir system is more desirable than the single reservoir problem due to its ability in controlling various environment parameters such as flooding. In these systems, the inflow of downstream reservoirs are affected by the outflow of their upstream reservoirs. In the OR literature, this case is considered much more complex and for the sake of simplicity mainly the single case in considered. For multi-reservoirs the main problem that leads to approximations to DP methods or sampling approaches is the curse of dimensionality (Mahootchi 2009). Our approach handles this complexity using continuous states and actions.

The objective in this example is to find the optimal time for draining the reservoirs so that maximum reward is obtained. If draining occurs too frequently (in small time frames), flooding may occur from the excess water. This means draining should occur as latest as possible. On the other hand, not draining will cause overflow of the reservoir and waste of the energy.

The MDP solution should solve a policy that obtains the maximum profit of electricity charges, while staying in the safe water levels of both reservoirs. In order to consider the time as a continuous variable, we allowing the system to choose to drain at any time. The choice of not draining will not gain any profit as we reward according to the electricity discharged. Two discrete actions of drain and no-drain is considered where the former is a of a continuous-time nature. The transition function is demonstrated below:

$$x_1' = 400 * e + x1 - 700 * e + 500 * e$$
$$x_2' = 400 * e + x2 - 500 * e$$

Here we take draining as the act of discharging water levels per time-step from the upper-stream reservoir to the down-stream reservoir ($500 * e$). A constant amount of discharge is always considered for the down-stream to ensure all electricity demands are fulfilled. The amount of rain (r) is considered as a constant(400)which affects both reservoirs at the time of discharge.

The reward function for both actions considers the safe range of [50,4500] as the safe water levels and assigns a positive reward of e for the action of draining, and no rewards (but also no penalty) for not draining. If the next state is not in the safe range, a huge penalty of -1+e6 is assigned as the reward:

$$(x_1 \leq 4500 - 200 * e) \wedge (x2 \leq 4500 + 100 * e)$$
$$\wedge (x_1 \geq 50 - 200 * e) \wedge (x2 \geq 50 + 100 * e) : e$$

We have presented the results for this experiment next. Because of the two actions and the fact that one is continuous, we would expect to see a draining action in one iteration and a no-drain action in the next iteration, since two draining could be performed in one longer step.

The first iteration starts with a drain only policy and achieves the value according to the current water levels of both reservoirs. The up-stream reservoir obtains the maximum value function if it has water levels above half of the safe range. The value decreases as water levels in the down-stream reservoir goes higher due to flood prevention.

We have demonstrated the optimal policy for the next two consequent iterations. In the second iteration, CSA-MDP will not drain the water because of the continuous nature of the problem, the required draining was performed in the previous iteration. As the water levels increase,

Related Work

Related Work

The most relevant vein of Related work is that of (Feng et al. 2004) and (Li and Littman 2005) which can perform exact dynamic programming on DC-MDPs with rectangular piecewise linear reward and transition functions that are delta functions. While SDP can solve these same problems, it removes both the rectangularity and piecewise restrictions on the reward and value functions, while retaining exactness. Heuristic search approaches with formal guarantees like HAO* (Meuleau et al. 2009) are an attractive future extension of SDP; in fact HAO* currently uses the method of (Feng et al. 2004), which could be directly replaced with SDP. While (Penberthy and Weld 1994) has considered general piecewise functions with linear boundaries (and in fact, we borrow our linear pruning approach from this paper), this work only applied to fully deterministic settings, not DC-MDPs.

Other work has analyzed limited DC-MDPS having only one continuous variable. Clearly rectangular restrictions are meaningless with only one continuous variable, so it is not surprising that more progress has been made in this restricted setting. One continuous variable can be useful for optimal solutions to time-dependent MDPs (TMDPs) (Boyan and Littman 2001). Or phase transitions can be used to arbitrarily approximate one-dimensional continuous distributions leading to a bounded approximation approach for arbitrary single continuous variable DC-MDPs (Marecki, Koenig, and Tambe 2007). While this work cannot handle arbitrary stochastic noise in its continuous distribution, it does exactly solve DC-MDPs with multiple continuous dimensions.

Finally, there are a number of general DC-MDP approximation approaches that use approximate linear programming (Kveton, Hauskrecht, and Guestrin 2006) or sampling

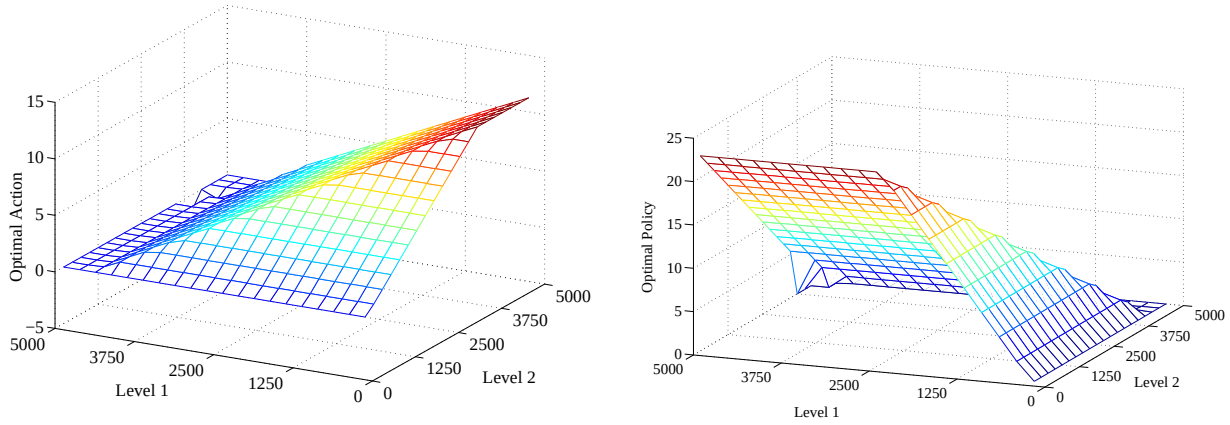

Figure 4: Optimal policy for iteration 2 and 3

in a reinforcement learning style approach (Remi Munos 2002). In general, while approximation methods are quite promising in practice for DC-MDPS, the objective of this paper was to push the boundaries of *exact* solutions; however, in some sense, we believe that more expressive exact solutions may also inform better approximations, e.g., by allowing the use of data structures with non-rectangular piecewise partitions that allow higher fidelity approximations.

Concluding Remarks

References

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.

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.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *National Conference on Artificial Intelligence AAAI*, 1010–1015.

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)*.

Zhang, N. L., and Poole, D. 1996. Exploiting causal independence in bayesian network inference. *J. Artif. Intell. Res. (JAIR)* 5:301–328.