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 *symbolically* solving a linear program 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 $d \in [-10, 10]$. 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 to indicate whether the picture has already been taken, x' and b' to denote post-action state, and R to denote reward, we can express this simple MARS ROVER CSA-MDP using piecewise dynamics and reward:

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

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

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

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.

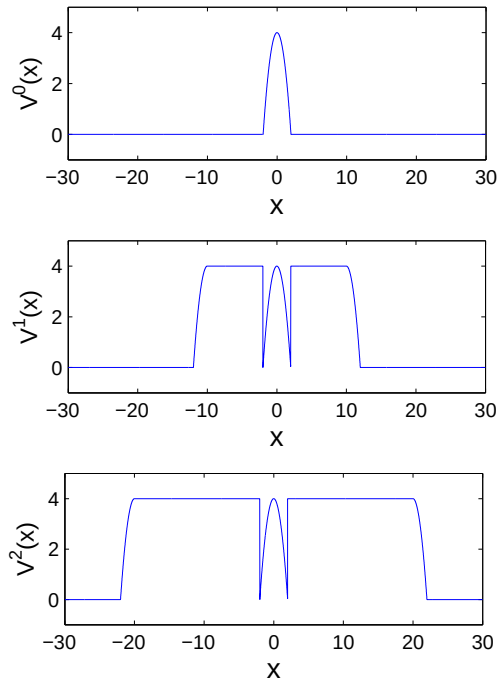

Figure 1: Optimal sum of rewards (value functions) $V^t(x)$ for $b = false$ for time horizons (i.e., decision stages remaining) $t = 0$, $t = 1$, and $t = 2$ on the MARS ROVER problem. For $x \in [-2, 2]$, the rover automatically takes a picture and receives a reward quadratic in x. For $V^1(x)$, the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

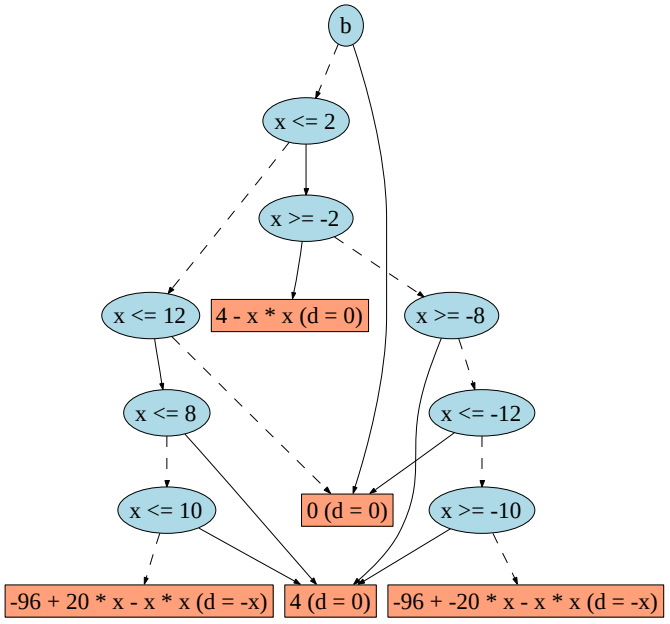

Figure 2: Optimal value function $V^1(x)$ for the MARS ROVER problem represented as an extended algebraic decision diagram (XADD). Here the solid lines represent the *true* branch for the decision and the dashed lines the *false* branch. To evaluate $V^1(x)$ for any state x, one simply traverses the 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* ($d = \pi^{*,1}(x)$) 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 *symbolically* solving a linear program 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

We first introduce general continuous state and action Markov decision processes (CSA-MDPs) and then our specific restricted version of CSA-MDPs with piecewise linear continuous transitions, arbitrary stochastic discrete transitions, and quadratic (or lower-order) reward. We then provide an algorithm their finite-horizon solution via symbolic dynamic programming by extending (Sanner, Delgado, and de Barros 2011).

Factored Representation

In a CSA-MDP, 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 state variable b_i ($1 \le i \le n$) is boolean s.t. $b_i \in \{0, 1\}$ and each x_j ($1 \le j \le m$) is continuous s.t. $x_j \in [L_j, U_j]$ for $L_j, U_j \in \mathbb{R}; L_j \le U_j$. We also assume a finite set of p actions $A = \{a_1(\vec{y}_1), \ldots, a_p(\vec{y}_p)\}$, where the $\vec{y}$ denote *continuous action parameters* for the corresponding action.

A CSA-MDP is defined by the following: (1) a 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 $a(\vec{y})$; (2) 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 (3) a dis-

count factor γ, $0 \leq \gamma \leq 1$.[2] 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 horizon-dependent policies $\Pi^* = (\pi^{*,1}, \ldots, \pi^{*,H})$ that maximizes the expected sum of discounted rewards over a horizon $h \in H; H \geq 0$:[3]

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

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 individual conditional probabilities $P(b'_i|\cdots)$ and $P(x'_j|\cdots)$ for each next state variable condition on the action and (a subset of) the variables in the action, current and next state. For simplicity of exposition, we assume there are no *synchronic arcs* (variables that condition on each other in the same time slice) within the binary $\vec{b}$ and continuous variables $\vec{x}$, but we allow synchronic arcs from variables in $\vec{b}$ to variables in $\vec{x}$.[4] We write the joint transition model as

$$P(\vec{b}', \vec{x}'|\vec{b}, \vec{x}, a, \vec{y}) = \quad (5)$$
$$\prod_{i=1}^{n} P(b'_i|\vec{b}, \vec{x}, a, \vec{y}) \prod_{j=1}^{m} P(x'_j|\vec{b}, \vec{b}', \vec{x}, a, \vec{y}).$$

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: (1) PLEs are first-order *Markov*, meaning that they can only condition on the previous state, (2) PLEs are deterministic, and (2) 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 Dirac $\delta[\cdot]$ 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 $x' = \{\ldots$ as a probability distribution, which is required in general when PLEs are used to encode the $P(x'_j|\vec{b}, \vec{b}', \vec{x}, a, \vec{y})$ as in this work.

It is important at this point to qualify the extent of stochasticity permitted in this restriction of CSA-MDPs. While it will be clear that our restrictions do not permit general stochastic transition noise (e.g., Gaussian white noise as in LQR), 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 heavily in previous exact continuous state MDP solutions (Feng et al. 2004; Meuleau et al. 2009).

We allow the reward function $R_a(\vec{b}, \vec{x}, \vec{y})$ to be either (1) a general piecewise linear function (boolean or linear conditions and linear valuations) such as

$$R_a(\vec{b}, \vec{x}, \vec{y}) = \begin{cases} x_1 \leq x_2 + 1: & 1 - x_1 + 2x_2 \\ x_1 > x_2 + 1: & 3x_1 + 2x_2 \end{cases} \quad (6)$$

or (2) a piecewise quadratic function restricted to be quadratic in one state variable and linear in *** such as (3). In the concluding remarks, we discuss the implications of relaxing the above restrictions on the transition and reward functions. For now, with our CSA-MDP defined as above, we proceed to discuss how to solve for it's optimal finite horizon value function and policy.

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 $V^0(\vec{b}, \vec{x})$ (e.g., to $V^0(\vec{b}, \vec{x}) = 0$) we define the quality of taking action a in state $(\vec{b}, \vec{x})$ and acting so as to obtain $V^h(\vec{b}, \vec{x})$ thereafter as the following:

$$Q_a^{h+1}(\vec{b}, \vec{x}) = \max_{\vec{y}} \left(R_a(\vec{b}, \vec{x}) + \gamma \cdot \right. \quad (7)$$
$$\left. \sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b'_i|\vec{b}, \vec{x}, a, \vec{y}) \prod_{j=1}^{m} P(x'_j|\vec{b}, \vec{b}', \vec{x}, a, \vec{y}) \right) V^h(\vec{b}', \vec{x}') d\vec{x}' \right)$$

Given $Q_a^h(\vec{b}, \vec{x})$ for each $a \in A$, we can proceed to define the $h+1$-stage-to-go value function as follows:

$$V^{h+1}(\vec{b}, \vec{x}) = \max_{a \in A} \left\{ Q_a^{h+1}(\vec{b}, \vec{x}) \right\} \quad (8)$$

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 $\pi^{*,h}(\vec{b}, \vec{x}) = \arg\max_a Q_a^h(\vec{b}, \vec{x})$. If the horizon $H = \infty$ and the optimal policy has finitely bounded value, then value iteration can terminate at horizon $h+1$ if $V^{h+1} = V^h$; then $\pi^*(\vec{b}, \vec{x}) = \arg\max_a Q_a^{h+1}(\vec{b}, \vec{x})$.

Of course this is simply the *mathematical* definition, next we show each of these algebraic operations can be computed for CSA-MDPs.

[2] If time is explicitly included as one of the continuous state variables, $\gamma = 1$ is typically used, unless discounting by horizon (different from the state variable time) is still intended.

[3] $H = \infty$ is allowed if an optimal policy has a finitely bounded value (guaranteed if $\gamma < 1$); for $H = \infty$, the optimal policy is independent of horizon, i.e., $\forall h \geq 0, \pi^{*,h} = \pi^{*,h+1}$.

[4] Synchronic arcs between variables within $\vec{b}$ or within $\vec{x}$ can be accommodated if the forthcoming Algorithm 2 (`Regress`) is modified to eliminate child variables before parent variables in the directed acyclic graph (DAG) formed by the DBN.

Algorithm 1: `Solve`(CSA-MDP, H)

begin
  $V^0 := 0, h := 0$
  while $h < H$ **do**
    $h := h + 1$
    foreach $a(\vec{y}) \in A$ **do**
      $Q_a^h(\vec{y}) :=$ `Regress` $(V^{h-1}, a, \vec{y})$
      $Q_a^h := \max_{\vec{y}} Q_a^h(\vec{y})$ *// Continuous* max
      $V^h := \max_a Q_a^h$ *// Discrete* max
      $\pi^{*,h} := \arg\max_{(a,\vec{y})} Q_a^h(\vec{y})$
    if $V^h = V^{h-1}$ **then**
      break *// Terminate if early convergence*
  return V^h
end

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

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

Symbolic Dynamic Programming (SDP)

In this section, we extend the symbolic dynamic programming (SDP) work of (Boutilier, Reiter, and Price 2001; Sanner, Delgado, and de Barros 2011) to the case of continuously parameterized actions for CSA-MDPs. For completeness, we recap the general SDP framework, noting that step 4 represents the major modification to prior SDP work.

Before we define our SDP solution, however, we must formally define our case representation and symbolic case operators.

Case Representation and Operators

Throughout this paper, we will assume that all symbolic functions can be represented in *case* form as follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \tag{9}$$

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 (a) boolean variables in $\vec{b}$ and (b) inequalities $(\geq, >, \leq, <)$, equalities $(=)$, or disequalities $(\neq)$ where the left and right operands can be *any* function of one or more variables in $\vec{x}$. 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 state assignments. The f_i can be *any* functions of the state variables in $\vec{x}$.

Unary operations such as scalar multiplication $c \cdot f$ (for some constant $c \in \mathbb{R}$) or negation $-f$ on case statements f are straightforward; the unary operation is 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:

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

One can verify that the resulting case statement is still within the case language defined previously. At first glance this may seem like a cheat and little is gained by this symbolic sleight of hand. However, simply having a case partition representation that is closed under maximization will facilitate the closed-form regression step that we need for SDP. Furthermore, the XADD that we introduce later will be able to exploit the internal decision structure of this maximization to represent it much more compactly.

The next operation of *restriction* is fairly simple: in this operation, we want to restrict a function f to apply only in cases that satisfy some formula ϕ, which we write as $f|_\phi$. This can be done by simply appending ϕ to each case partition as follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \qquad f|_\phi = \begin{cases} \phi_1 \wedge \phi : & f_1 \\ \vdots & \vdots \\ \phi_k \wedge \phi : & f_k \end{cases}$$

Clearly $f|_\phi$ only applies when ϕ holds and is undefined otherwise, hence $f|_\phi$ is a partial function unless $\phi \equiv \top$.

The final operation that we need to define for case statements is substitution. *Symbolic substitution* simply takes a set σ of variables and their substitutions, e.g., $\sigma = \{x_1'/(x_1 + x_2), x_2'/x_1\}$ where the LHS of the $/$ represents the substitution variable and the RHS of the $/$ represents the expression that should be substituted in its place. No variable occurring in any RHS expression of σ can also occur in any LHS expression of σ. We write the substitution of a non-case function f_i with σ as $f_i\sigma$; as an example, for the σ defined previously and $f_i = x_1' + x_2'$ then $f_i\sigma = x_1 + x_2 + x_1$ as would be expected. We can also substitute into case partitions ϕ_j by applying σ to each inequality operand; as an example, if $\phi_j \equiv x_1' \leq 2x_2'$ then $\phi_j\sigma \equiv x_1 + x_2 \leq 2x_1$. Having now defined substitution of σ for non-case functions f_i and case partitions ϕ_j we can define it for case statements in general:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \qquad f\sigma = \begin{cases} \phi_1\sigma : & f_1\sigma \\ \vdots & \vdots \\ \phi_k\sigma : & f_k\sigma \end{cases}$$

One property of substitution is that if f has mutually exclusive partitions ϕ_i $(1 \leq i \leq k)$ then $f\sigma$ must also have mutually exclusive partitions — this follows from the logical consequence that if $\phi_1 \wedge \phi_2 \models \bot$ then $\phi_1\sigma \wedge \phi_2\sigma \models \bot$.

Maximization of Continuous Parameters

In the previous section, we covered the operations required for the SDP algorithm defined in (Sanner, Delgado, and de Barros 2011). As the primary technical contribution of this paper, we now address the issue of maximizing a function (e.g., Q-function) w.r.t. continuous parameters (e.g., action parameters).

It is perhaps easiest to demonstrate how to compute $\max_d f(d)$ in exact, closed-form for $f(d)$ in case format with a running example. If $f(d)$ has mutually exclusive and exhaustive case partitions w.r.t. a fixed d (a property that will be guaranteed whenever this operator is applied in SDP), the following sequence of equalities hold:

$$\max_d f(d) = \max_d \sum_i \phi_i(d) f_i(d) \qquad = \max_d \max_i \phi_i(d) f_i(d)$$
$$= \max_i \max_d \phi_i(d) f_i(d).$$

The critical step where $\sum_i$ is replaced with $\max_i$ is permitted since the ϕ_i are mutually disjoint and exhaustive — only one ϕ_i will ever be true for any given assignment to the case statement variables. This step is crucial because the outer $\max_i$ is the "case maximization" operation already discussed while the inner $\max_d$ is a continuous variable maximization that we can now compute for just one case partition, i.e., one $\phi_i(d)$ and $f_i(d)$, and the apply a case maximization to all of the results for each partition.

To continue with maximization in one partition, we use our MARS ROVER NONLINEARexample in one of the case partitions of the first iteration for continuous maximization. This partition i (resulting from applying SDP) contains conditions $\phi_i(d) \equiv \neg b \wedge (x \geq 2) \wedge (d \leq 10) \wedge (d \geq -10) \wedge (x + d \leq 2) \wedge (x + d \geq -2)$ and value $f_i(d) = 4 - (x + d)^2$.

To perform $\max_d \phi_i f_i(d)$, we consider some mathematical definitions necessary for the correctness of this operation:

1. A real-valued function f defined on the real domain is said to have a global (or absolute) maximum point at the point x, if for all x: $f(x^*) \geq f(x)$. The value of the function at this point is the maximum of the function.
2. By *Fermat's theorem*, the local maxima (and minima) of a function can occur only at its critical points: where either the function is not differentiable (which only happens at the upper and lower boundaries) or its first derivative is 0.
3. The *Boundedness theorem* states that a continuous function f in the closed interval (a,b) is bounded on that interval. That is, there exist real numbers m and M such that:

$$\forall x \in \{a, b\} : m \leq f(x) \leq M$$

Based on these definitions, the maximum value of $\phi_i f_i$ must occur at either the boundaries or the points where its first derivative is zero. Because we only consider up to quadratic f_i in this paper, we need only consider the single root of f_i if it is quadratic. We can write the explicit functions for these bounds in *piecewise linear case form* for the lower bounds LB given by ϕ_i (i.e., $d \geq -10$, $d \geq x - 2$), upper bounds UB given by ϕ_i (i.e., $d \leq 10$, $d \leq 2 - x$), and the roots of $\frac{\partial}{\partial d} f_i = -2x - 2d = 0$ w.r.t. d (i.e., $d = -x$). Since we have multiple lower and upper bounds, we must consider the maximum of the lower bounds and the minimum of the upper bounds which we can represent in case form as follows:

$$LB := \max(-10, -2 - x) = \begin{cases} x \leq 8 : & -2 - x \\ x \geq 8 : & -10 \end{cases} \quad UB := \min(10, 2 - x) =$$

We know that the maximum of the f_i could occur at any of the above points, but we do not know which. Hence the true maximum is simply the case maximization of the substitution of the UB, LB, and root into $f(d)$.

A lower bound on action d occurs when the inequalities of $\geq, >$ have action d on their LHS; then any expression on the RHS is considered a lower bound on action d. Similar to this, an upper bound occurs for inequalities of $\leq, <$ and the RHS expression is considered an upper bound for action d. The roots of the function are also defined by taking the first derivative of f with respect to d and setting it to zero.

To enforce correctness of bounds symbolically we must have $LB \leq Roots \leq UB$, so we add the pair of constraints for the bounds to the root constraint:

$$\begin{cases} -10 \leq -x \\ -2 - x \leq -x \\ -x \leq 10 \\ -x \leq 2 - x \end{cases} = \begin{cases} x \leq 10 \\ x \geq -10 \end{cases}$$

The other two statements can be removed as a tautology, to leave the final root partition as:

$$Root := \begin{cases} x \leq 10 \wedge x \geq -10 : & -x \\ \neg(x \leq 10 \wedge x \geq -10) : & 0 \end{cases}$$

Having the UB,LB and root of each case partition, we can maximize the value of each case function according to the bounds. We want to factor out the action variable d (we show later that this is equal to maximizing the Q-function values in order to obtain the value function). This means replacing the continuous action with the possible maximum points; LB, UB and roots. Replacing the action with a constant, variable or another function is equal to applying the *substitute* operator. Each substitution forms a partition of the final maximization over the three posibble maximum points:

$$\max_d \begin{cases} 4-(x+UB)^2 \\ 4-(x+LB)^2 \\ 4-(x+Root)^2 \end{cases}$$

We take the maximum of the UB and LB substitutions and then maximum the result with the root substitution. Below shows the substituted maximum points:

$$\begin{cases} x \le -8: & -96+20\times x-x^2 \\ x \ge -8: & 0 \end{cases}$$
$$\begin{cases} x \le 8: & 0 \\ x \ge 8: & -96-x^2-20\times x \end{cases}$$
$$\begin{cases} x \le 10 \wedge x \ge -10: & 4 \\ x \le 10 \wedge x \ge -10: & 4 \end{cases})$$

The main partitions after the maximization over action d is as below, the rest of the partitions hold the value of zero:

$$\max_d \begin{cases} -10 \le x \le 10 : 4 \\ 10 \le x \le 12 \wedge -96-x^2-20\times x \ge 0 : \\ -96-x^2-20\times x \ge 0 \\ -12 \le x \le -10 \wedge -96-x^2+20\times x \ge 0 : \\ -96-x^2+20\times x \ge 0 \end{cases}$$

After taking the symbolic maximum over the partitions above, we also add all the constraints that did not effect the d boundaries (i.e, $x \le 2 : false$ and $b : false$) which means only considering the partition values for $x \ge 2$ which takes out the final branch of the case partition above.

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 (**?**), 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.

Symbolic Dynamic Programming (SDP)

In the SDP solution for CSA-MDPs, our objective will be to take a CSA-MDP as defined in Section **??**, apply value iteration as defined in Section , and produce the final value optimal function V^h at horizon h in the form of a case statement.

For the base case of $h = 0$, we note that setting $V^0(\vec{b},\vec{x}) = 0$ (or to the reward case statement, if not action dependent) is trivially in the form of a case statement.

Next, $h > 0$ requires the application of SDP. Fortunately, given our previously defined operations, SDP is straightforward and can be divided into four steps:

1. *Prime the Value Function*: Since V^h will become the "next state" in value iteration, we setup a substitution $\sigma = \{b_1/b_1', \ldots, b_n/b_n', x_1/x_1', \ldots, x_m/x_m'\}$ and obtain $V'^h = V^h\sigma$.
2. *Continuous Integration*: Now that we have our primed value function V'^h in case statement format defined over next state variables $(\vec{b}', \vec{x}')$, we first evaluate the integral marginalization $\int_{\vec{x}'}$ over the continuous variables in (7). Because the lower and upper integration bounds are respectively $-\infty$ and ∞ and we have disallowed synchronic arcs between variables in $\vec{x}'$ in the transition DBN, we can marginalize out each x_j' independently, and in any order. Using *variable elimination* (Zhang and Poole 1996), when marginalizing over x_j' we can factor out any functions independent of x_j' — that is, for $\int_{x_j'}$ in (7), one can see that initially, the only functions that can include x_j' are V'^h and $P(x_j'|\vec{b},\vec{b}',\vec{x},a,\vec{y}) = \delta[x_j' - g(\vec{x})]$; hence, the first marginal over x_j' need only be computed over $\delta[x_j' - g(\vec{x})]V'^h$.
   The key insight of the SDP algorithm introduced in (Sanner, Delgado, and de Barros 2011) is that the integration $\int_{x_j'} \delta[x_j' - g(\vec{x})]V'^h dx_j'$ simply *triggers the substitution* $\sigma = \{x_j'/g(\vec{x})\}$ on V'^h, that is
   $$\int_{x_j'} \delta[x_j' - g(\vec{x})]V'^h dx_j' \;=\; V'^h\{x_j'/g(\vec{x})\}. \qquad (10)$$

Thus we can perform the operation in (10) repeatedly in sequence *for each* x'_j $(1 \leq j \leq m)$ for every action a.

Hence to perform (10) on this more general representation, we obtain that $\int_{x'_j} P(x'_j|\vec{b}, \vec{x}, a, \vec{y}) V'^h dx'_j$

$$= \begin{cases} \phi_1 : & V'^h\{x'_j = f_1\} \\ \vdots & \vdots \\ \phi_k : & V'^h\{x'_j = f_k\} \end{cases}.$$

This operation may be complicated in some cases for when the form of $P(x'_j|\vec{b}, \vec{x}, a, \vec{y})$ is a *conditional* equation, but the result is still in a closed-form case statement as outlined in (Sanner, Delgado, and de Barros 2011).

To perform the full continuous integration, if we initialize $\tilde{Q}_a^{h+1} := V'^h$ for each action $a \in A$, and repeat the above integrals for all x'_j, updating $\tilde{Q}_a^{h+1}$ each time, then after elimination of all x'_j $(1 \leq j \leq m)$, we will have the partial regression of V'^h for the continuous variables for each action a denoted by $\tilde{Q}_a^{h+1}$.

3. *Discrete Marginalization*: Now that we have our partial regression $\tilde{Q}_a^{h+1}$ for each action a, we proceed to derive the full backup Q_a^{h+1} from $\tilde{Q}_a^{h+1}$ by evaluating the discrete marginalization $\sum_{\vec{b}'}$ in (7). Because we previously disallowed synchronic arcs between the variables in $\vec{b}'$ in the transition DBN, we can sum out each variable b'_i $(1 \leq i \leq n)$ independently. Hence, initializing $Q_a^{h+1} := \tilde{Q}_a^{h+1}$ we perform the discrete regression by applying the following iterative process *for each* b'_i in any order for each action a:

$$Q_a^{h+1} := \left[Q_a^{h+1} \otimes P(b'_i|\vec{b}, \vec{x}, a, \vec{y}) \right] |_{b'_i=1} \oplus \left[Q_a^{h+1} \otimes P(b'_i|\vec{b}, \vec{x}, a, \vec{y}) \right] |_{b'_i=0}. \quad (11)$$

This requires a variant of the earlier restriction operator $|_v$ that actually *sets* the variable v to the given value if present. Note that both Q_a^{h+1} and $P(b'_i|\vec{b}, \vec{x}, a, \vec{y})$ can be represented as case statements (discrete CPTs *are* case statements), and each operation produces a case statement. Thus, once this process is complete, we have marginalized over all $\vec{b}'$ and Q_a^{h+1} is the symbolic representation of the intended Q-function.

4. *Continuous Parameter Maximization*: Now that we have $Q_a^{h+1}(\vec{y})$ in the case format with free parameters $\vec{y}$ for the action variables, we simply need to compute $Q_a^{h+1} = \max_{\vec{y}} Q_a^{h+1}(\vec{y})$. This continuous variable maximization can be done sequentially for each element of $\vec{y}$ as outlined previously. At the same time, if record the maximal substitutions $\vec{y}$ made at each leaf, we can also recover the parameterized policy that led to these optimal values, e.g., as shown in Figure 2.

5. *Maximization*: Now that we have Q_a^{h+1} in case format for each action $a \in \{a_1, \ldots, a_p\}$, obtaining V^{h+1} in case format as defined in (8) requires sequentially applying *symbolic maximization* as defined previously:

$$V^{h+1} = \max(Q_{a_1}^{h+1}, \max(\ldots, \max(Q_{a_{p-1}}^{h+1}, Q_{a_p}^{h+1})))$$

By induction, because V^0 is a case statement and applying SDP to V^h in case statement form produces V^{h+1} in case statement form, we have achieved our intended objective with SDP. On the issue of correctness, we note that each operation above simply implements one of the dynamic programming operations in (7) or (8), so correctness simply follows from verifying (a) that each case operation produces the correct result and that (b) each case operation is applied in the correct sequence as defined in (7) or (8).

Empirical Results

We implemented the SDP algorithm using the XADDs 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 subsections we will study these examples empirically.

INVENTORY CONTROL

This domain problem is a well-known optimization benchmark in the OR literature. Business firms often deal with the problem of deciding upon the amount of product to order in a time period so that the customer demands are satisfied. The firm's warehouse will keep an inventory of this product to address different customer demand levels. Each month, the firm must decide on the amount of products to order based on the current stock level. The order should not be too high since keeping the inventory is expensive, nor should it be too low in which case it will be penalized for being unable to meet customer demands and leading to the loss of customers. The optimization problem faced by the firm is to find an optimal order policy that maximizes the profit. (Mahootchi 2009)

We present a simple formulation of this problem where the capacity of the inventory is C units of each product and customer orders not satisfied in this month are backlogged for the next month, so inventory can take negative values. We consider two cases, a one product inventory with one order action and the other with two products that needs two different orders.

We take two continuous state variable $x_1, x_2 \in [-1000, C]$ indicating the current inventory quantity into account, with the total inventory capacity of 800, 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 $a_1, a_2 \in [0, C]$ which can at most take the value of the maximum inventory capacity.

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 also a positive reward for fulfilling the customer demand whenever there are enough stocks in the inventory. The transition for one of the state variables and the reward function are defined below: current stock level, leaving it negative customer orders iteration

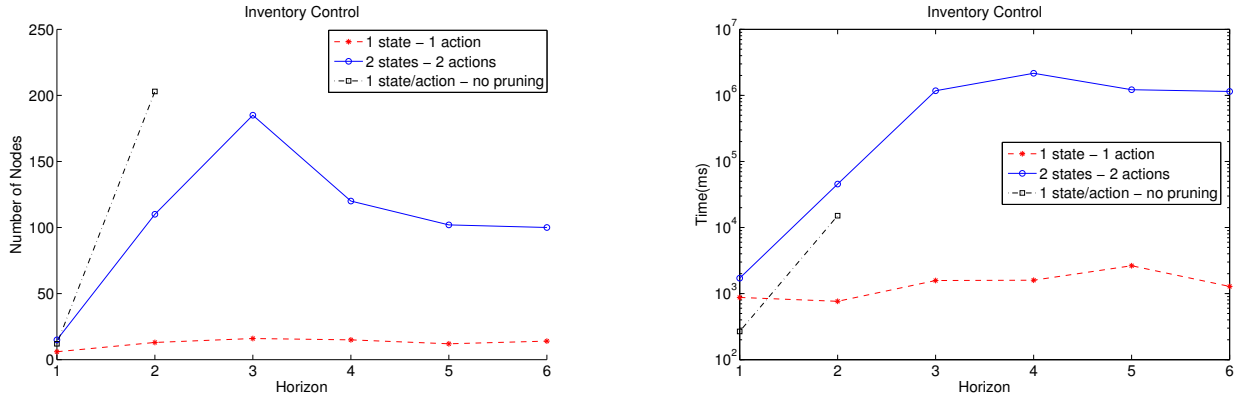

Figure 3: Space (# XADD nodes in value function) and time for different horizons of CSA-MDP on INVENTORY CONTROL Comparing 1D, 2D and no-pruning

$$x_1' = \begin{cases} d \wedge (x_1 + a_1 + x_2 - 150 \leq 800): & x_1 + a_1 - 150 \\ d \wedge (x_1 + a_1 + x_2 - 150 \geq 800): & x_1 - 150 \\ \neg d \wedge (x_1 + a_1 + x_2 - 150 \leq 800): & x_1 + a_1 - 50 \\ \neg d \wedge (x_1 + a_1 + x_2 - 150 \geq 800): & x_1 - 50 \end{cases}$$

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

$$R = \begin{cases} (x_1 + x_2 \geq 900) \wedge d \\ 150 - 0.5 \times a_1 - 0.4 \times a_2 - 0.1 \times (x_1 + x_2) \\ (x_1 + x_2 \leq 900) \wedge d \\ (150 - (x_1 + x_2)) - 0.5 \times a_1 - 0.4 \times a_2 - 0.1 \times (x_1 + x_2) \\ (x_1 + x_2 \geq 300) \wedge \neg d \\ 50 - 0.5 \times a_1 - 0.4 \times a_2 - 0.1 \times (x_1 + x_2) \\ (x_1 + x_2 \leq 300) \wedge \neg d \\ (50 - (x_1 + x_2)) - 0.5 \times a_1 - 0.4 \times a_2 - 0.1 \times (x_1 + x_2) \end{cases}$$

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 and the reward function is based on the demand levels and the current stock in inventory. If the current stock is larger than the total inventory, we get the reward for fulfilling the demand $(e.g 150)$, if the demand is high and the inventory is not high enough, then the reward is $(e.g(150 - (x_1 + x_2)))$, in both cases the action costs and holding costs are also 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 a 1-product INVENTORY CONTROLproblem with a multi-dimensional one. Figure 3 compares the time and nodes for different iterations for these two problem instances and a third comparisoon for the effect of not pruning on the 1D instance. This demonstrates the impact of having multiple constraints and action variables on the problem size which requires much more state-action partitions. 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. 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.

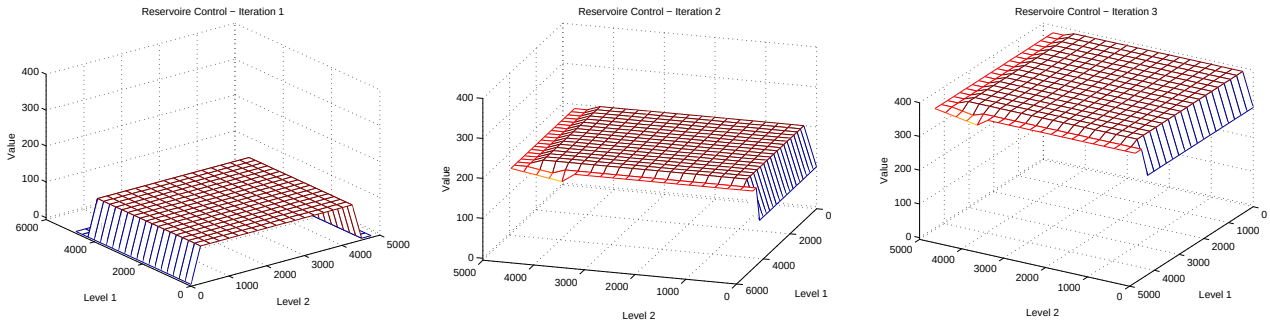

Figure 4: Exact optimal value function for RESERVOIR MANAGEMENTdomain

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.

In figure 5, both Q-values of drain and no-drain for the second iteration have been visualized. As expected the value function follows from selecting the no-drain action. The Q-value of draining is similar to that of the first iteration and the final value function is similar to the Q-function of no-drain. The difference of the recent two value functions is in the fact that since draining was performed in the first iteration, in the second iteration, the down-stream reservoir is not in danger of flooding and can obtain the optimal value (22.5). Since flooding is a direct result of excess water in x_1, the optimal value is for the lower range levels in the down-stream compared to the levels of x_2.

Further iterations of the exact CSA-DP algorithm results in more partitions on the state-action space. The optimal policy converges to the value obtained in the second iteration. There are some minor difference in the value of each iteration due to the fact that with higher horizons, the reservoir can plan to obtain higher rewards and prevent flooding or overflowing. We provide the Q-values of draining, not draining and the value function for iteration 9 in 6 to show this effect. Also from this plot we want to confirm the fact that in the 9^th iteration, the policy should choose to drain (as all other odd-iterations). The difference between the Q-values of the two actions is very minor. The sum of distances between the value function and the two Q-values shows that this distance is 49.4887 for the drain action while it is equal to 90.5312 for the no-drain action. This shows that despite the small difference, the policy tends more towards draining in this iteration.

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

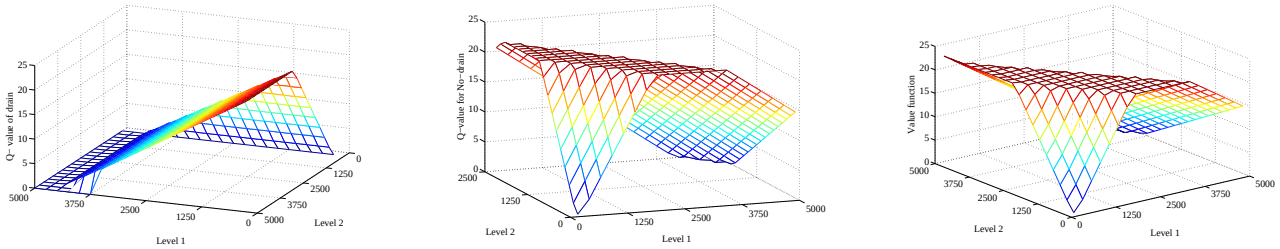

Figure 5: Value function and Q-function values of actions: Drain, No-Drain for second iteration

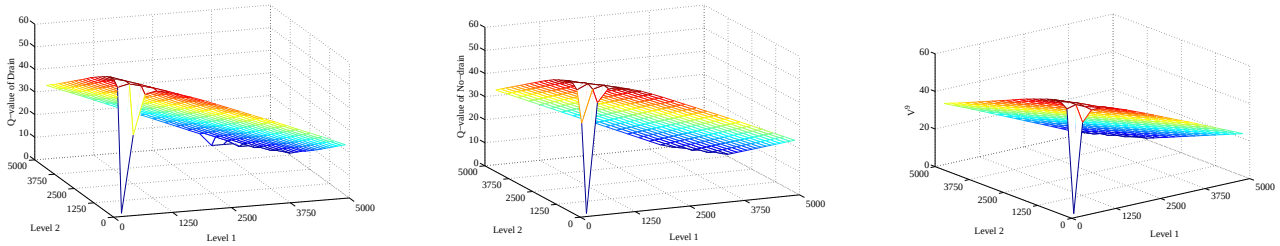

Figure 6: Value function and Q-function values of actions: Drain, No-Drain for the 9th iteration

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 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.