Exact Solutions to 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. Previous work on such problems has only provided exact solutions in limited settings, leaving one to resort to discretization or sampling as approximate solution approaches. In this work, we propose a symbolic dynamic programming solution to obtain the *optimal closed-form* value function and policy for CSA-MDPs with discrete noise, piecewise linear dynamics, and piecewise quadratic rewards. Crucially we show how the continuous action maximization step in the dynamic programming backup can be evaluated optimally and symbolically for this case; further we extend this maximization operation to work with an efficient and compact data structure, the extended algebraic decision diagram (XADD). We demonstrate empirical results for CSA-MDPs on a range of domains from planning and operations research to demonstrate the first *exact solution* to these problems — even in the case of multivariate actions and nonlinear reward.

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 variant of the MARS ROVER problem (Bresina et al. 2002), a rover must navigate within a continuous environment and carry out assigned scientific discovery tasks. In oversubscribed cases where the MARS ROVER may not be able to visit all of its assigned landmarks (e.g., to take pictures) within a given time horizon, it may still be rewarded to some degree for partial objective fulfillment (e.g, taking pictures within the vicinity of the assigned landmarks).

Previous work on *exact* solutions to 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 additive noise in a partially observable setting. However, the transition dynamics and reward (alternately cost) of such problems are not allowed to be piecewise — a restriction that prevents us from solving problems like the aforementioned MARS ROVER problem.

To be concrete about the modeling power and contributions of this paper, let us formalize a MARS ROVER problem to serve as a running example:

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 an arbitrary continuous distance $d \in [-10, 10]$. The rover receives its greatest reward for snapping 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; we use boolean variable b to indicate whether the picture has already been taken. Formally, using x' and b' to denote the post-action state and R to denote the reward function, we obtain a simple instance of a CSA-MDP:*[1]

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

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

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

There are two natural questions that we want to ask in CSA-MDP settings such as this one:

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

(b) What is the corresponding policy one should execute to achieve this optimal value?

To get a sense of the form of the optimal solution to problems such as the 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 we see that

[1] One will note that this CSA-MDP example is deterministic for purposes of minimal exposition; the solution in the paper will generally allow for CSA-MDPs with discrete noise.

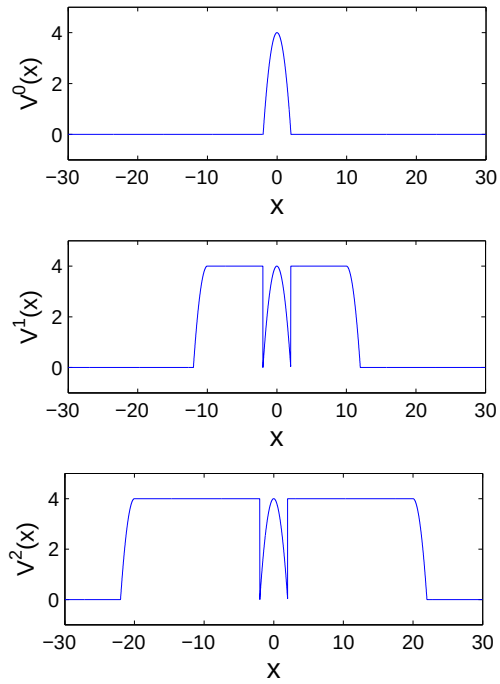

Figure 1: Optimal value functions V^t (for $b = \mathit{false}$) for time horizons (i.e., decision stages remaining) $t = 0$, $t = 1$, and $t = 2$ on the MARS ROVER problem. For $x \in [-2, 2]$, the rover automatically takes a picture and receives a reward quadratic in x. For V^1, the rover may move up to 10 units in either direction, reaching the full reward of 4 up to $x = \pm 10$ and non-zero reward up to $x = \pm 12$. For V^2, the rover can move up to 20 units in two time steps, allowing it to achieve non-zero reward up to $x = \pm 12$.

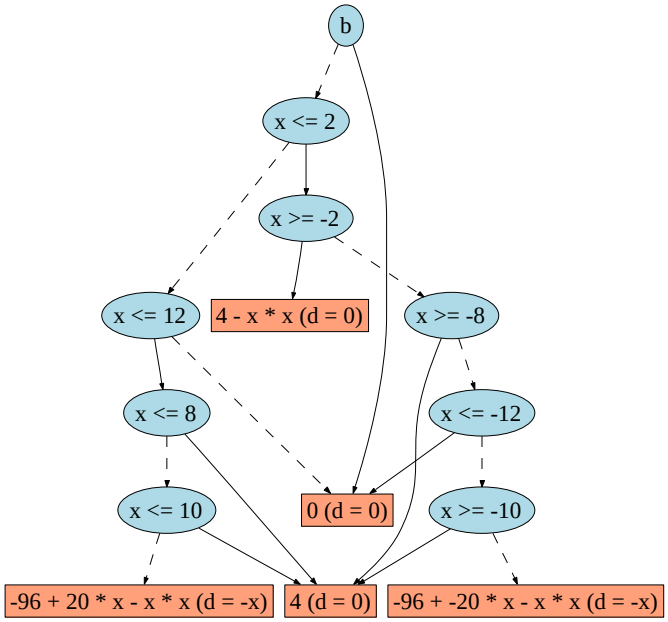

Figure 2: Optimal value function V^1 for the MARS ROVER problem represented as an extended algebraic decision diagram (XADD). Here the solid lines represent the *true* branch for the decision and the dashed lines the *false* branch. To evaluate $V^1(x)$ for any state x, one simply traverses the diagram in a decision-tree like fashion until a leaf is reached where the expression provides the value. The second expression in parentheses in each leaf provides the optimal action *policy* for d as a function of the state that allows one to obtain $V^1(x)$. This closed-form policy can be derived as a byproduct of symbolic dynamic programming as we discuss later.

the piecewise nature of the transition and reward function lead to piecewise structure in the value function and policy. And despite their striking simplicity, we are not aware of any exact solution method to the above MARS ROVER variant that can produce such an optimal closed-form result.

To this end, we extend the symbolic dynamic programming (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 CSA-MDPs with discrete noise, piecewise linear dynamics, and piecewise quadratic rewards. 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 and how it can be efficiently realized in the extended algebraic decision diagram (XADD) we use to perform all SDP operations. This allows us to obtain the *first* algorithm to derive exact closed-form solutions to this class of CSA-MDPs along with a closed-form representation of the optimal policy (cf. Figure 2). We empirically evaluate the time and space required to compute exact solutions to the MARS ROVER problem for different time horizons, as well as WATER RESERVOIR and multivariate action INVENTORY CONTROL studied in operations research showing the first exact solutions for these problems can be practically computed for reasonable horizons.

Continuous State and Action MDPs

We first introduce continuous state and action Markov decision processes (CSA-MDPs) and then review their finite-horizon solution via dynamic programming building on (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 (defined below) and action a; (2) a reward function $R(\vec{b}, \vec{x}, a, \vec{y})$, which specifies the immediate reward obtained by taking action a in state $(\vec{b}, \vec{x})$; and (3) a discount factor γ, $0 \le \gamma \le 1$.[2] A policy π specifies

[2]If time is explicitly included as one of the continuous state variables, $\gamma = 1$ is typically used, unless discounting by horizon

the action $\pi(\vec{b}, \vec{x})$ 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]

$$V^{\Pi^*}(\vec{x}) = E_{\Pi^*}\left[\sum_{h=0}^{H} \gamma^h \cdot r^h \Big| \vec{b}_0, \vec{x}_0\right], \tag{1}$$

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 $(\vec{b}, \vec{x})$; 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 $P(b_i' | \cdots, a)$ and $P(x_j' | \cdots, a)$ condition on a subset of the variables in the current and next state. We disallow *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 $\vec{b}$ to $\vec{x}$ (note that these conditions enforce the directed acyclic graph requirements of DBNs). We write the joint transition model as

$$P(\vec{b}', \vec{x}' | \vec{b}, \vec{x}, a, \vec{y}) = \prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a, \vec{y}) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y}) \tag{2}$$

where $P(b_i' | \vec{b}, \vec{x}, a, \vec{y})$ may condition on a subset of $\vec{b}$, $\vec{x}$, and $\vec{y}$ and likewise $P(x_j' | \vec{b}, \vec{b}', \vec{x}, a, \vec{y})$ may condition on a subset of $\vec{b}$, $\vec{b}'$, $\vec{x}$, and $\vec{y}$.

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. 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 *conditional stochastic linear equations* (CSLEs). For the solution provided here, we only require three properties of these CSLEs: (1) they are *Markov*, meaning that they can only condition on the previous state, (2) they are piecewise linear (pieces entering through the conditional part) and (2) they are *deterministic* meaning that the next state must be uniquely determined from the previous state (i.e., $x_1' = x_1 + x_2^2$ is deterministic whereas $x_1'^2 = x_1^2$ is not because $x_1' = \pm x_1$).[4] Otherwise, we allow for arbitrary functions in these Markovian, conditional deterministic equations as in the following example:

$$P(x_1' | \vec{b}, \vec{b}', \vec{x}, a) = \delta\left[x_1' - \begin{cases} b_1' \wedge x_2^2 \leq 1: & \exp(x_1^2 - x_2^2) \\ \neg b_1' \vee x_2^2 > 1: & x_1 + x_2 \end{cases}\right] \tag{3}$$

Here the use of the Dirac $\delta[\cdot]$ function ensures that this is a conditional probability function that integrates to 1 over x_1' in this case. But in more intuitive terms, one can see that this $\delta[\cdot]$ encodes the deterministic transition equation $x_1' = \ldots$ where $\ldots$ is the conditional portion of (3). In this work, we require all CSLEs in the transition function for variable x_i' to use the $\delta[\cdot]$ as shown in this example.

It will be obvious that CSLEs in the form of (3) are *conditional linear equations*; they are furthermore *stochastic* because they can condition on boolean random variables in the same time slice that are stochastically sampled, e.g., b_1' in (3). Of course, these CSLEs are restricted in that they cannot represent general stochastic noise (e.g., Gaussian noise), but 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; Li and Littman 2005; Meuleau et al. 2009).

We allow the reward function $R_a(\vec{b}, \vec{x})$ to be a piecewise quadratic function of the current state for each action $a \in A$ with parameters $\vec{y}$, for example:

$$R_a(\vec{b}, \vec{x}, \vec{y}) = \begin{cases} x_1^2 + x_2^2 \leq 1: & 1 - x_1^2 - x_2^2 \\ x_1^2 + x_2^2 > 1: & 0 \end{cases} \tag{4}$$

or even

$$R_a(\vec{b}, \vec{x}, \vec{y}) = 10 x_3 x_4 \exp(x_1^2 + \sqrt{x_2}) \tag{5}$$

With our CSA-MDP now completely defined, our next objective is to solve it.

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:

$$Q_a^{h+1}(\vec{b}, \vec{x}) = R_a(\vec{b}, \vec{x}) + \gamma \cdot \sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a)\right) V^h(\vec{b}', \vec{x}') d\vec{x}' \tag{6}$$

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:

$$V^{h+1}(\vec{b}, \vec{x}) = \max_{a \in A} \left\{Q_a^{h+1}(\vec{b}, \vec{x})\right\} \tag{7}$$

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 $\pi^{*,h}(\vec{b}, \vec{x}) = \arg\max_a Q_a^h(\vec{b}, \vec{x})$. If the

(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] While the *deterministic* requirement may seem to conflict with the label of *stochastic*, we note that stochasticity enters through the conditional component to be discussed in a moment.

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. In the discrete-only case, we can always compute this in tabular form; however, how to compute this for DC-MDPs with reward and transition function as previously defined is the objective of the symbolic dynamic programming algorithm that we define next.

Symbolic Dynamic Programming

As it's name suggests, symbolic dynamic programming (SDP) (**?**) is simply the process of performing dynamic programming (in this case value iteration) via symbolic manipulation. While SDP as defined in (**?**) was previously only used with piecewise constant functions, we now generalize the representation to work with general piecewise functions needed for DC-MDPs in this paper.

Before we define our 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:

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

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 $\vec{x}$.

As concrete examples, consider the transition representation for KNAPSACKin Ex. , the optimal value function for KNAPSACKfrom (**??**), or any of (3), (4), or (5).

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 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., $\sigma = \{x_1'/(x_1 + x_2), x_2'/x_1^2 \exp(x_2)\}$ 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 $f_i\sigma = x_1 + x_2 + x_1^2 \exp(x_2)$ as would be expected. We can also substitute into case partitions ϕ_j by applying σ to each inequality operand; as an example, if $\phi_j \equiv x_1' \leq \exp(x_2')$ then $\phi_j\sigma \equiv x_1 + x_2 \leq \exp(x_1^2 \exp(x_2))$. 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 actions

In the previous section, we covered the operations required for the SDP algorithm. Here we address the issue of maximizing a function (e.g., Q-function) over continuous action variables.

To compute the maximum of any arbitrary function f in the case format, over the continuous variable d, we maintain a closed-form case representation. Using the mutually exclusive and disjoint property of partitions, :[5] we claim that the following equality holds:

$$\sum_d \max_i \varphi_i f_i = \max_i \max_d \varphi_i f_i \tag{8}$$

Since we care about taking the maximum of each partition and obtaining an overall maximum for the function and the partitions have no junctions, the above holds for our case representation. Thus we compute the maximum separately for each case partition and then $\max$ over the final results. The final maximum is a symbolic maximum as explained before, the inside maximum for each partition is the operation defined here.

To continue with maximization in one partition, we use our MARS ROVER NONLINEARexample in one of the branches of the first iteration for continuous maximization that covers this technique completely. This branch is obtained from replacing the transition function of each variable in the reward function for the conditions $\neg b \wedge (x \geq 2) \wedge (d \leq 10) \wedge (d \geq -10) \wedge (x + d \leq 2) \wedge (x + d \geq -2)$ to obtain the root value of $4 - (x + a)^2$.

We also consider some mathematical definitions for the technique of maximizing the continuous action:

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 or its derivative is 0 at that point.
3. 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 a function is defined at the boundaries and roots of that function according to the continuous variable d. If this function were linear and a polynomial of a certain degree k, then the maximum for that function would occur on any of the (k-1) roots of the function or at the lower and upper bounds defined for it.

We can write the explicit functions for these bounds in *piecewise linear case form* for the respective lower and upper bounds LB, UB and possible roots of our example as shown below. The maximum of all the lower bounds is obtained by taking a maximum over this set and the maximum of the upper bounds is obtained by taking a minimum over the set of UB.

$$LB := \max \begin{cases} d \geq -10 \\ d \geq -2 - x \end{cases} = \begin{cases} x \leq 8 : & -2 - x \\ x \geq 8 : & -10 \end{cases}$$

$$UB := \max \begin{cases} d \leq 10 \\ d \leq 2 - x \end{cases} = \begin{cases} x \geq -8 : & 2 - x \\ x \leq -8 : & 10 \end{cases}$$

$$Roots := \left\{ -2 \cdot x - 2 \cdot d = 0 := -x \right.$$

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:

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

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:

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

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:

[5] Any case partition without this property can be converted into its equivalent case representation using logical operators such as $\wedge$ and $\neg$

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

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

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

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.

Symbolic Dynamic Programming (SDP)

In the SDP solution for DC-MDPs, our objective will be to take a DC-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 (6). 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 (6), 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 $\delta[x'_j - g(\vec{x})]V'^h$.
   What follows is one of the *key novel insights of SDP* in the context of DC-MDPs — 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

$$\int_{x'_j} \delta[x'_j - g(\vec{x})]V'^h dx'_j \;=\; V'^h\{x'_j/g(\vec{x})\}. \quad (9)$$

   Thus we can perform the operation in (9) repeatedly in sequence *for each* x'_j $(1 \le j \le m)$ for every action a. The only additional complication is that the form of $P(x'_j|\vec{b}, \vec{x}, a, \vec{y})$ is a *conditional* equation, c.f. (3), and represented generically as follows:

$$P(x'_j|\vec{b}, \vec{x}, a, \vec{y}) = \delta\left[x'_j = \begin{cases} \phi_1: & f_1 \\ \vdots & \vdots \\ \phi_k: & f_k \end{cases}\right] \quad (10)$$

   Hence to perform (9) on this more general representation, we obtain that $\int_{x'_j} P(x'_j|\vec{b}, \vec{x}, a, \vec{y})V'^h dx'_j$

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

   In effect, we can read (10) as a *conditional substitution*, i.e., in each of the different *previous state* conditions ϕ_i $(1 \le i \le k)$, we obtain a different substitution for x'_j appearing in V'^h (i.e., $\sigma = \{x'_j/f_i\}$). Here we note that because V'^h is *already* a case statement, we can simply replace the single partition ϕ_i with the multiple partitions of $V'^h\{x'_j/f_i\}|_{\phi_i}$.[6] This reduces the *nested* case statement back down to a non-nested case statement as in the following example:

$$\begin{cases} \phi_1: & \begin{cases} \psi_1: & f_{11} \\ \psi_2: & f_{12} \end{cases} \\ \phi_2: & \begin{cases} \psi_1: & f_{21} \\ \psi_2: & f_{22} \end{cases} \end{cases} = \begin{cases} \phi_1 \wedge \psi_1: & f_{11} \\ \phi_1 \wedge \psi_2: & f_{12} \\ \phi_2 \wedge \psi_1: & f_{21} \\ \phi_2 \wedge \psi_2: & f_{22} \end{cases}$$

   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 $(1 \le j \le m)$, 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 (6). Because we previously disallowed synchronic arcs between the variables in $\vec{b}'$ in the transition DBN, we can sum out each variable b'_i $(1 \le i \le n)$ 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

[6] If V'^h had mutually disjoint partitions then we note the restriction and substitution operations preserve this disjointness.

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. it Continuous Maximization Now that we have Q_a^{h+1} in the case format we perform continuous maximization for the continuous parameter y as explained in the previous section. This step factors out the continuous action variable from the Q-function.

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 (7) 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 (6) or (7), 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 (6) or (7).

On a final note, we observe that SDP holds for *any* symbolic case statements; we have not restricted ourselves to rectangular piecewise functions, piecewise linear functions, or even piecewise polynomial functions. As the SDP solution is purely symbolic, SDP applies to *any* DC-MDPs using bounded symbolic function that can be written in case format! Of course, that is the theory, next we meet practice.

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), we extend this formalism to handle continuous variables in a data structure we refer to as the XADD. An example XADD for the optimal MARS ROVER NONLINEARvalue function for horizon 2 is provided in Figure 3.

In brief we note that an XADD is like an ADD except that (a) the decision nodes can have arbitrary inequalities, equalities, or disequalities (one per node) and (b) the leaf nodes can represent arbitrary functions. The decision nodes still have a fixed order from root to leaf and the standard ADD operations to build a canonical ADD (REDUCE) and to perform a binary operation on two ADDs (APPLY) still applies in the case of XADDs.

While exact solutions using symbolic dynamic programming are possible in principle for arbitrary symbolic CSE transition and reward functions, we note that it is much more difficult to devise a canonical and compact form for representations such as (5) in comparison to (4). Hence while we have used general examples throughout the paper to demonstrate the expressiveness of our approach, we will restrict XADDs to use *polynomial* functions only. We note the main advantage of this for the XADD is that we can put the leaf and decision nodes in a *unique, canonical* form, which allows us to minimize redundancy in the XADD representation of a case statement.

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 is irrelevant, although this 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, we can use the feasibility checkers of a linear programming solver (e.g., as also done in (Penberthy and Weld 1994)) to prune unreachable nodes in the XADD; later we show results demonstrating impressive reductions in XADD size using this style of pruning.

The only two XADD operations that pose difficulty are substitution and maximization. In principle substitution is simple, the only caveat is that substitutions modify the decision nodes and hence decision nodes may become unordered. We can use the recursive application of ADD binary operations $\otimes$ and $\oplus$ as given in Algorithm 1 to correctly reorder the nodes in an XADD F after substitution. A related reordering issue occurs during XADD maximization; because XADD maximization can introduce new decision nodes (which occurs at the leaf when two leaf functions are compared) and these decision nodes may be out of order w.r.t. the diagram, reordering as defined in Algorithm 1 must also be applied after maximization.

On a final note, we mention that an implementation of SDP using case statements without any attempt to merge and simplify cases often cannot get past the first or second SDP iteration; as our results show next, XADDs allow SDP to scale to much longer horizons in practice.

Empirical Results

We implemented the CSA-DP algorithm using the XADDs and tested it on several domains. The first example which has been used throughout the paper is the MARS ROVER NONLINEAR, problems from the OR literature; INVENTORY CONTROLand WATER RESERVOIR. In the following subsections we will study these examples empirically.

MARS ROVER NONLINEAR

The problem specifications has already been defined in the previous sections.

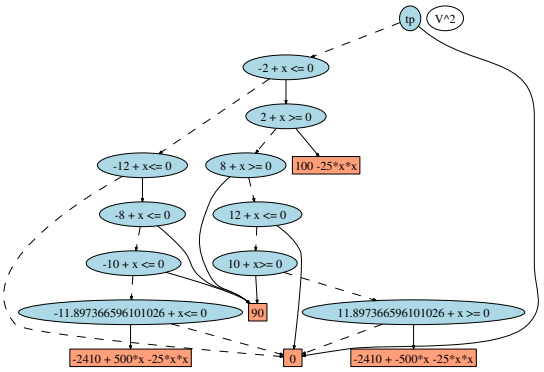

Figure 3: The final value function for the second horizon of the MARS ROVER NONLINEARexample

Algorithm 1: REORDER(F)

input : F (root node for possibly unordered XADD)
output: F_r (root node for an ordered XADD)
begin
  //if terminal node, return canonical terminal node
  if *F is terminal node* **then**
    return *canonical terminal node for polynomial of F*;
  //nodes have a $true$ & $false$ branch and var id
  if *$F \rightarrow F_r$ is not in Cache* **then**
    F_{true} = REORDER (F_{true}) $\otimes \mathbb{I}[F_{var}]$;
    F_{false} = REORDER (F_{false}) $\otimes \mathbb{I}[\neg F_{var}]$;
    $F_r = F_{true} \oplus F_{false}$;
    insert $F \rightarrow F_r$ in Cache;
  return F_r;
end

This plot visualizes non-linear piecewise boundaries which occurs due to non-linear leaves in the XADD ;equivalent to non-linear partitions of the state space. It shows that the Rover can move non-linearly (quadratic) around the target radius (i.e, [-2,2]) progressively to get to the exact point with high precision. It then achieves a constant optimal value after the radius and before reaching the action boundaries in each iteration (i.e, the rover can move at most between [-10,10] in each iteration). After the boundaries the rover moves according to a non-linear function thus achieving a non-linear value which was demonstrated in Figure **??**.

Every step of the algorithm allows the Rover to move from further distances towards the radius, hence the constant step and the non-linear curvature at the boundaries occur iteratively. This example clearly demonstrated the power of the CSA-DP algorithm for non-linear functions.

We note that these results were generated with the XADD pruning which consists of linear pruning using the LP solver and non-linear pruning using the linearization method. Without the pruning, even in low horizons, because of infeasible branches, time and space grows exponentially. We have also used the general pruning method of XADDs as in (Sanner, Delgado, and de Barros 2011) for the other domain problems.

INVENTORY CONTROL

This domain problem has become a benchmark problem for optimization in the OR literature and applies to our general DCA-MDP setting. Business firms often deal with the problem of deciding the amount of product to order in a time period such that customer demands are satisfied. The firm's warehouse will keep an inventory of this product to deal with 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 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 reward function are defined below:

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

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

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

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 $(e.g150)$, 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 4 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 and as the figure shows, time and space grow non-linearly after the second iteration.

WATER RESERVOIR

The problem of WATER RESERVOIRneeds 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 effected 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 using other simplifications is the curse of dimensionality. In this domain the discharge action is considered as a discrete action and so are the energy demands. This causes the state space to grow exponentially in case of multiple states and actions. (Lamond and Boukhtouta 2002) Using our method for continuous action value iteration, we show that this problem can be handled efficiently and is scalable to multi-reservoir problems. Consider a two-level reservoir with the outflow of the second reservoir added to the input of the first reservoir. The state space is the level of water in both reservoirs as well as the energy demands and inflow (such as rainfall or streams) to the reservoirs $(l1, l2, i, e)$. We consider the water levels as continuous variables, the inflows to both reservoirs are the same which depends on a high-low sessions of rainfall. The energy demands e can be considered both continuous or discrete for different customer needs.

We want to prevent the upstream reservoir $(l2)$ from reaching low water levels, while avoiding flooding which is caused by high water levels. A constant amount of water is discharged from the downstream reservoir$(l1)$ to meet customer electricity demands. The inflow to both reservoirs is assumed to be the same (same amount of rainfall).

$$l1' = \begin{cases} d: & l1 - 1000 \cdot e + 500 \cdot e + 450 \cdot e \\ \neg d: & l1 - 1000 \cdot e + 500 \cdot e + 250 \cdot e \end{cases}$$

$$l2' = \begin{cases} d: & l2 - 500 \cdot e + 450 \cdot e \\ \neg d: & l2 - 500 \cdot e + 250 \cdot e \end{cases}$$

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

$$R = \begin{cases} l1 \leq 4500 \wedge l2 \leq 4500 \wedge l1 \geq 50 : e \\ l1 \leq 4500 \wedge l2 \leq 4500 \wedge l1 \leq 50 : -100 + e \\ \neg(l1 \leq 4500 \wedge l2 \leq 4500) : 0 \end{cases}$$

The reward is assigned based on preventing the flood. If water levels prevent flooding, a reward is given for the *time* the reservoir selects the action of discharging water from one reservoir to the other$(l2tol1)$, or the choice of not discharging at that time, depending on the water levels. The elapsed time is used as the reward value if the constraints are met, in this case the system should choose to perform an action (drain or no drain) as latest as possible. This elapsed time is added to the current time at each transition. The water levels are transitioned based on the previous levels and the amount of discharge and inflow multiplied by the elapsed time. This

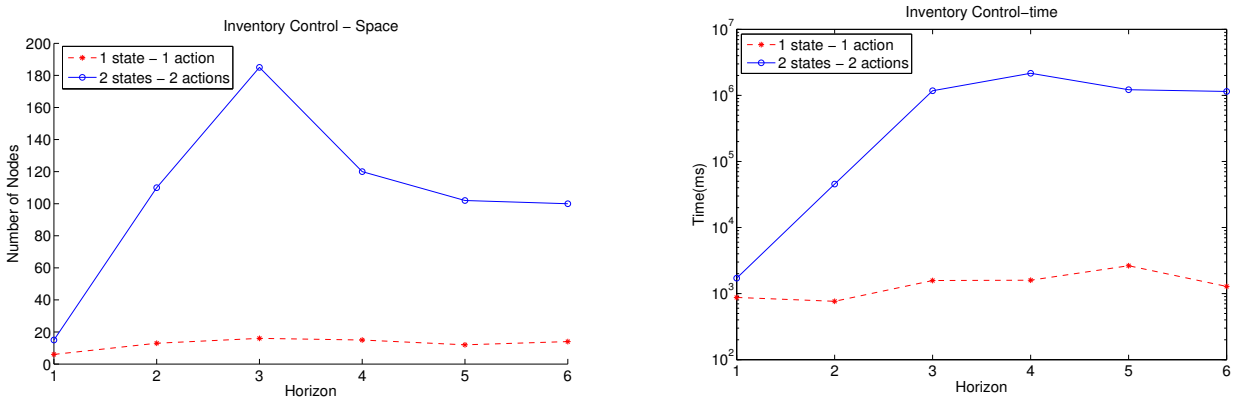

Figure 4: Space (# XADD nodes in value function) and time for different horizons of CSA-DP on INVENTORY CONTROL Comparing 1D, 2D and no-pruning

means that higher elapsed time is desired to maximize the reward, while lower ones ensure flood prevention.

In order to demonstrate the effectiveness of planning optimally with our model, we use this continuous-time example of the multi-reservoir where the elapsed time is defined as the action parameter to drain a reservoir. The goal is to show a form of value function refinement that is generated using this CSA-DP algorithm. The optimal value function will obtain higher values for higher horizons, while the value for lower horizons were less than the current value function. This is demonstrated in Figure 5 where three value functions have been presented in different horizons.

Here flood prevention is ensured in each step by the linear piecewise boundaries on the very high and very low levels, the action of discharging occurs here either at time 0 for very critical points or at a time depending on the water levels later during the iteration (i.e, $a = -8.18 + 0.0018 * l1$ if one of the water levels is above the critical level in the third iteration). As for the constant section, this piece is the safe range and the action of discharge (or not discharge) is left till the latest possible time thus achieving higher rewards. As the iteration number increases, the future discounted reward goes higher than the previous horizon.

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 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.

Conclusions

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.

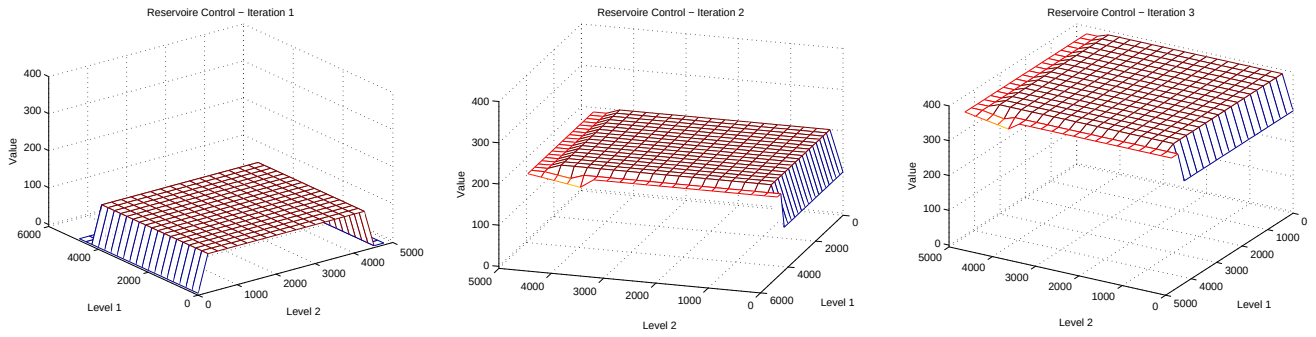

Figure 5: Exact optimal value function for WATER RESERVOIRdomain

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

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.

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.