Symbolic Dynamic Programming for Continuous action MDPs

Zahra Zamani
NICTA & the ANU
Canberra, Australia
zahra.zamani@anu.edu.au

Scott Sanner
NICTA & the ANU
Canberra, Australia
ssanner@nicta.com.au

Cheng Feng
Department of Aeronautics and
Astronautics, MIT, USA
cfang@mit.edu

Abstract

Many real-world decision-theoretic problems require continuous actions as well as continuous states in order to plan optimally. While previous work has used methods such as discretization or sampling to approximately solve such problems, no exact solution has been introduced for both continuous state and action spaces. In this work, we take a symbolic approach to modelling domain problems with discrete and continuous state and action Markov decision processes (MDPs) and propose a symbolic dynamic programming solution to obtain the optimal policy. Crucially we show how that continuous maximization step in the dynamic programming backup can be evaluated optimally and symbolically; 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 continuous state and action MDPs on a range of domains from planning and operations research to demonstrate the first exact optimal solution to these problems.

1 Introduction

Markov decision processes (MDPs)[] have been a theoretical model for many planning problems in the recent years. Although the most common solution to MDPs is to use value iteration or policy iteration, they can quickly fall into Bellman's curse of dimensionality problem for large state action spaces. In many problems this can be avoided by modelling the problem with continuous state action spaces which is also a closer resemblance to real-world problems. Examples of problems in the continuous state and action domain can include Robotics, Operation Research and Management problems. The OR (Operation Research) literature has used MDPs to model many problems such as Inventory Control [] or Water Reservoir management []. These are examples where the state space can be modelled by discrete and continuous Markov decision processes and where the actions can also be considered continuous.

Most existing solutions concentrate on approximation techniques such as sampling [] that requires computational effort while providing no guarantee to the optimal solution, or other methods such as discretizing the state-action space[] that has scalability problems for large domain sizes. Meanwhile little effort has been devoted to exact solutions using the continuous variables beyond the piecewise rectilinear value function setting which contributes to a small subset of the problems in this domain.

In the previous paper [], the solution to continuous state MDPs was investigated, here we introduce a new model for Discrete and continuous state and Action MDPs (DCA-MDPs) and propose a new algorithm for value iteration in a Symbolic Dynamic Programming(SDP) format. Although the exact solution presented in [] covers the discrete and continuous state domain, it could not be used for planning in continuous action spaces.

We exploit the algebraic properties of the maximization operator for continuous actions and develop disjoint case partitions that can be used to factor the actions in the Q-value to obtain the next value function. This symbolic maximization approach allows us to go beyond the linear function domain and apply the technique for non-linear reward and transition functions. Our method is scalable to higher horizons as well as being applicable to multi-variate continuous action domains. We represent the reward function as the function that is used to reward and penalize the agent based on the action costs. Both the reward and the transition function are represented using First-order logic with arbitrary piecewise symbol functions. These will be explained more deeply in section 2. This section also explains the main technique for applying continuous actions to any arbitrary function in depth.

The practical implementation of case statements of section 2 is the extended ADDs (XADDs) structure, which is discussed in section 3. In section 4 the symbolic value iteration algorithm for continuous action selection is presented and the approach of maximizing the action variables is ex-

amined by visual examples. Empirical results on various problems such as the Mars Rover, Inventory control and Water reservoirs are then demonstrated in section 5. We present some of the related works in section 6 and conclude with a discussion on the future work in the last section.

2 Case Representation

To work in the symbolic space with lifted solutions, we need a representation for the symbolic functions and logical operators. Here we define such a notation along with the required operations for the SDP algorithm in the next sections. We consider the following simple example that is used throughout this paper:

[INVENTORY CONTROL] 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. Inventory control has become a benchmark problem for optimization in the OR literature and applies to our general domain settings. Here we present a simple formulation of this problem:

Example (INVENTORY CONTROL)**.** *We consider order decisions and delivery at the beginning of each month and order satisfaction to take place at the end of the month. The capacity of the inventory is C units of the product and customer orders not satisfied in this month are backlogged for the next month, so inventory can take negative values. Here we have a continuous state variable $x \in [-1000, C]$ indicating the current inventory quantity, with C=500, 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 $a \in [0, C]$ which can at most take the value of the maximum inventory capacity.*

The *case* notation can present any symbolic function in the following format:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \tag{1}$$

Where the ϕ_i are logical formulae defined over the state and action space $(\vec{d}, \vec{x}, \vec{a})$ that can include arbitrary logical $(\wedge, \vee, \neg)$ combinations of the boolean variable $\vec{d}$ and can also contain any sort of inequalities (such as $\geq, >, \leq, <$) where the left and right operands can be *any* function of the continuous variables $\vec{x}, \vec{a}$. The f_i can be *any* functions of the state action variables.

Each of the case statements introduce a partitioning on the domain where each partition is mutually exclusive and disjoint from the other partitions since otherwise they can be converted into more partitions with this property using logical operators:

$$\begin{cases} \\ \varphi_1 : f_1 \\ \varphi_2 : f_2 \\ \\ \end{cases} \simeq \begin{cases} \varphi_1 \wedge \varphi_2 : mutal \\ \varphi_1 \wedge \neg\varphi_2 \\ \neg\varphi_1 \wedge \varphi_2 \\ \neg\varphi_1 \wedge \neg\varphi_2 : nondisjoint \end{cases}$$

The reward and transition functions below can show the case statements in more details:

Example (INVENTORY CONTROL)**.** *We define an immediate negative reward for the cost of producing an order (half of the order value) and the storage cost of holding the products in the inventory (one tenth of the total inventory stock) and also a positive reward for fulfilling the customer demand whenever there are enough stocks in the inventory. We can formalize the transition and reward for* INVENTORY CONTROL *the order action using conditional equations, where (x, d) and (x', d') are respectively the pre- and post-action state and R is immediate reward:*

$$x' = \begin{cases} d : & x + a - 150 \\ \neg d : & x + a - 50 \end{cases}$$

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

$$R = \begin{cases} x \geq 150 \wedge d : & 150 - 0.5 \cdot a - 0.1 \cdot x \\ x \geq 50 \wedge \neg d : & 50 - 0.5 \cdot a - 0.1 \cdot x \\ x \leq 150 \wedge d : & (150 - x) - 0.5 \cdot a - 0.1 \cdot x \\ x \leq 150 \wedge \neg d : & (50 - x) - 0.5 \cdot a - 0.1 \cdot x \end{cases}$$

Having defined the function notation in case statements, we now consider applying logical operators on a function. If the operator is *Unary* such as scalar multiplications (e.g. $0.5 \cdot a$) or negation $-f$ it is simply applied to each of the disjoint partitions.

For *Binary* operations, the cross-product of the logical partitions is considered and then the operator is applied to the result. The reward function of INVENTORY CONTROLis an example for binary $\oplus$ where the logical AND $(\wedge)$ of two logical formulas $x \geq 150$ and d is taken along with their negation, to form 4 partitions with different function definitions.

Likewise, $\ominus$ and $\otimes$ are applied by subtracting or multiplying partition values respectively. Apart from the logical

operators, we also need to perform other operations such as restriction, substitution and maximization on case statements to be able to completely explain the SDP algorithm.

In *restriction* a function Q is restricted to cases that satisfy some formula ϕ (e.g. the binary variable d_i then function can be restricted to either the true or false branch $Q|_{d_i}$). Restriction can be performed by appending the logical formula (e.g. of INVENTORY CONTROLsets $d_i = 0 or d_i = 1$) to each case partition:

$$Q = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \qquad Q|_{d_i=0} = \begin{cases} \phi_1 \wedge \neg d_i : & Q_1 \\ \vdots & \vdots \\ \phi_k \wedge \neg d_i : & Q_k \end{cases}$$

Symbolic substitution takes a set σ of variables and their substitutions, e.g.

$$\sigma = \{d_1/d'_1, \ldots, d_n/d'_n, x_1/x'_1, \ldots, x_m/x'_m\}$$

where the LHS of the $/$ represents the substitution variable and the RHS of the $/$ represents the expression that should be substituted in its place. While the example substitutes the next state (primed) with the current state, we can use substitution again on the transition functions of the variables to build the next state completely:

$$\sigma = \{d'/(d : 0.7, \neg d : 0.3), \\ x'/(d : x + a - 150; \neg d : x + a - 50)\}$$

For each partition, we perform substitution on the logical part ϕ and the functional part f. For the logical part of case partitions substitution occurs by applying σ to each inequality operand. For example if $\phi_i : d' \wedge x' > 200$ after substituting the result is $\phi_i\sigma : d \wedge x + a > 350$ where with d was chosen with probability 0.7.

As for the function values, they are again substituted in a similar fashion; if $f_i = 2x' + x'^2$ then $f_i\sigma = x^2 + a^2 + 2ax - 298x - 298a + 1950$. The partitions remain mutually exclusive and disjoint after the substitution due to logical consequence of applying σ. Substitution is defined generally in the following:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \qquad f\sigma = \begin{cases} \phi_1\sigma : & f_1\sigma \\ \vdots & \vdots \\ \phi_k\sigma : & f_k\sigma \end{cases} \tag{2}$$

Symbolic maximization is defined by considering all combinations of the logical formulas ϕ_i and ψ_i and taking the maximum according to the function values f and g:

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

According to this definition, maximization of any finite number of functions can be performed by taking a pair of the functions and iteratively building the resulting maximization. For the finite number of discrete actions the following maximization results in the value function:

$$V = \max(Q_{a_1}, \max(\ldots, \max(Q_{a_{p-1}}, Q_{a_p}))) \tag{3}$$

Apart from this maximization, we also have to consider maximizing a function (such as the Q-function) for continuous variables below.

2.1 Maximization of continuous actions

In the previous section, we covered the operations required for the SDP algorithm in section 4 except for maximizing a function (such as the Q-function) for an infinite number of actions (Δ) in our continuous case.

To compute a maximum of a function over the continuous variable Δ, $\max_\Delta f$ for f in the case format (1), we can rewrite it in the following equivalent form using the mutually exclusive and disjoint property of partitions (3):

$$\max_\Delta \sum_i \varphi_i f_i = \sum_i \max_\Delta \varphi_i f_i \tag{4}$$

Hence we can compute the maximum separately for each case partition (producing a case statement) and then $\sum$ the results using $\oplus$. To continue with maximization, we use an example of a continuous maximization with two case statements provided below that covers this technique completely:

$$\max_\Delta \begin{cases} (\Delta \geq 0) \wedge (x \leq 20) \wedge (\Delta \leq 3 * x) : & 99 - \Delta \\ (\Delta \leq 200) \wedge (x \geq 10) \wedge (x < \Delta - 5) : & -\Delta^2 + 9 \end{cases}$$

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. 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:

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

3. The first partial derivatives as to x (the variable to be maximized) are zero at the maximum. The second partial derivatives are negative. These are only necessary, not sufficient, conditions for a maximum because of the possibility of saddle points.

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 Δ. 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 and UB for each of the case statements respectively:

$$\begin{cases} LB := \Delta \geq 0 : & 0 \\ UB := \Delta \leq 3 * x : & 3 * x \\ ROOT := 99 - \Delta : & none \end{cases}$$

$$\begin{cases} LB := \Delta > x + 5 : & x + 5 \\ UB := \Delta \leq 200 : & 200 \\ ROOT := -\Delta^2 + 9 : & -3, +3 \end{cases}$$

A lower bound on action Δ occurs when the inequalities of $\geq, >$ have action Δ on their LHS; then any expression on the RHS is considered a lower bound on action Δ. Similar to this, an upper bound occurs for inequalities of $\leq, <$ and the RHS expression is considered an upper bound for action Δ. The roots of the function are also defined by taking the first derivative of f with respect to Δ.
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 Δ and have a function that is Δ-free (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 (2). The result of the maximum defines a case statement itself, which implies maximization over the two values obtained after the substitution of the LB and UB in the function definition of the case. The same is applied to the second case statement with the maximization to hold for the root function as well:

$$\begin{cases} \max_\Delta(99 - UB, 99 - LB) \\ \max_\Delta(-UB^2 + 9, -LB^2 + 9, (-ROOTS^2 + 9)) \end{cases}$$

$$\begin{cases} \max_\Delta(99 - 3x, 99 - 0) \\ \max_\Delta(-200^2 + 9, -(x + 5)^2 + 9, (9, 0) \end{cases}$$

The result is a normal symbolic maximization as defined in the previous section. To enforce correctness of bounds symbolically we must have $LB \leq ROOT \leq UB$, this

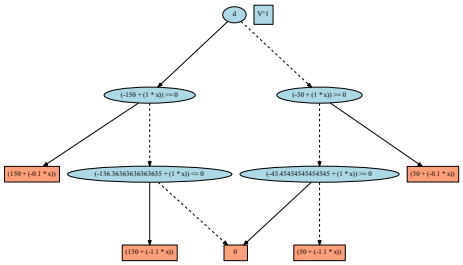

Figure 1: The optimal value function for horizon one of INVENTORY CONTROL as a decision diagram: the *true* branch is solid, the *false* branch is dashed.

means adding the pair of constraints for the bounds to the final result of maximization, for the first case this is $(3x \geq 0)$ and for the second equation:

$$(x + 5 \leq 200) \wedge (3 \geq x + 5) \wedge$$
$$(-3 \geq x + 5) \wedge (-3 \leq 200) \wedge (3 \leq 200)$$

The last two statements can be removed as a tautology, but the other constraints must remain.

3 Extended ADDs (XADDs)

In practice, a case statement representation of a value function with explicit partitions is computationally expensive. Extended Algebric decision diagrams (XADDs) were introduced in [] and will be used as the development tool throughout this work. XADDs were motivated by the SPUDD [**?**] algorithm which maintains compact value function representations for finite discrete factored MDPs using algebraic decision diagrams (ADDs) [**?**], and was extended to handle continuous variables. An example XADD for the optimal INVENTORY CONTROLvalue function is provided in Figures1 1. In an XADD the decision nodes can have arbitrary inequalities (one per node) and the leaf nodes can represent arbitrary functions. Standard ADD operations to build a canonical ADD and to perform a binary operation on two ADDs applies in the case of XADDs.

It is fairly straightforward for XADDs to support all case operations defined in the previous section. Binary operations or restriction is done similar to the ADD case while maximization and substitution requires reordering of the decision nodes since the former introduces new nodes and the latter modifies the decision nodes.

For the maximization of continuous variables, we consider two separate XADDs for the upper and lower bounds on the continuous action and then apply the maximum on these two functions. This operation is shown in detail in the next section.

In general, the fact that the decision nodes have internal

structure is irrelevant, but means that certain paths in the XADD may be inconsistent or infeasible (due to parent decisions). To further simplify our settings, when the XADD has only linear decision nodes, we can use the feasibility checkers of a linear programming solver to prune unreachable nodes in the XADD following the approach of []. Using the XADD structure the value iteration algorithm can scale to much longer horizons compared to the original SDP algorithm. In the next section we introduce the model we use for our SDP algorithm.

4 Discrete and Continuous State-Action MDPs

We assume that a general class of fully observable Markov decision processes with discrete and continuous states and actions can model our domain problems. The state space is represented by a vector $(\vec{b}, \vec{x}) = (b_1, \ldots, b_n, x_1, \ldots, x_m)$ consisting of Boolean variables b_i $(1 \leq i \leq n)$ such that $b_i \in \{0, 1\}$ and continuous Real variables x_j $(1 \leq j \leq m)$ s.t. $x_j \in [L_j, U_j]$ and $L_j, U_j \in \mathbb{R}; L_j \leq U_j$. The action space contains a finite number of continuous actions $A = \{a_1, \ldots, a_p\}$ where each $a_\Delta \in [L_\Delta, U_\Delta]$ and $L_\Delta, U_\Delta \in \mathbb{R}; L_\Delta \leq U_\Delta$.

The transition function is defined as the probability of the next state given the current state and action assuming the Markov property. Considering the Dynamic Bayesian Network model for modelling each state variable and disallowing synchronic arcs in between them, results in the following definition for the transition function:

$$P(\vec{b}', \vec{x}' | \vec{b}, \vec{x}, a_\Delta) = \prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a_\Delta) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a_\Delta) \tag{5}$$

Here the first probability for binary discrete variables can be represented by conditional probability tables (CPTs). The second probability for continuous variables is represented using conditional stochastic equations (CSEs) wthat are Markovian and deterministic with the definition of any arbitrary (e.g. non-linear) function with continuous actions and states such as:

$$P(x_1' | \vec{b}, \vec{b}', \vec{x}, a_\Delta) = \delta \left[x_1' - \begin{cases} b_1' \wedge a_\Delta \leq 1 : & \exp(x_1^2 - x_2^2) \\ \neg b_1' \vee x_2^2 > 1 : & x_1 + x_2 + a_\Delta \end{cases} \right] \tag{6}$$

Using the Dirac function the conditional probability is guaranteed to integrate to 1 over the variable, furthermore the probability is stochastic since it conditions on Boolean random variables as well as sampling them stochastically at the same time.

The reward function represents the immediate reward of taking an action in a state and it can take the form of any arbitrary function such as:

$$R_{a_\Delta}(\vec{b}, \vec{x}) = \begin{cases} x_1^2 + x_2^2 \leq 1 : & 1 - x_1^2 - x_2^2 \\ x_1^2 + x_2^2 > 1 : & 2 * a_\Delta^2 \end{cases} \tag{7}$$

The policy π specifies which action a_Δ to take in the current state $(\vec{b}, \vec{x})$ and the goal is to find an optimal sequence of policies $\Pi^* = (\pi^{*,1}, \ldots, \pi^{*,H})$ that maximizes the expected sum of discounted reward over a horizon $h \in H$:

$$V^{\Pi^*}(\vec{x}) = E_{\Pi^*} \left[\sum_{h=0}^{H} \gamma^h \cdot r^h \middle| \vec{b}_0, \vec{x}_0 \right], \tag{8}$$

where γ is the discount factor between 0 and 1, and r^h is the reward in horizon h assuming the state state is at $h = 0$. This optimal policy can be obtained using the value iteration algorithm. Starting with an initial value function, the value of taking an action in the current state is defined using the Q-function:

$$Q_{a_\Delta^{h+1}}(\vec{b}, \vec{x}) = R_{a_\Delta}(\vec{b}, \vec{x}) + \gamma \cdot \sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b_i' | \vec{b}, \vec{x}, a_\Delta) \prod_{j=1}^{m} P(x_j' | \vec{b}, \vec{b}', \vec{x}, a_\Delta) \right) V^h(\vec{b}', \vec{x}') d\vec{x}' \tag{9}$$

Now given the Q-function for each action, the h+1-stage-to-go value function is defined as (3):

$$V^{h+1}(\vec{b}, \vec{x}) = \max_{a \in A} \left\{ Q_a^{h+1}(\vec{b}, \vec{x}) \right\} \tag{10}$$

If the horizon is finite then the optimal value function continues to the last horizon but if the problem has infinite horizon then a termination criteria is defined, in most problems if the value function of two successive iterations are equal, the algorithm terminates. In the next section we define the symbolic dynamic programming algorithm as the solution to DCA-MDPs.

5 Symbolic Dynamic Programming (SDP)

In this section we present our exact algorithm for obtaining the optimal policy for continuous action and states. This algorithm avoids approximating as well as enumerating the state and action. The Continuous state-action dynamic programming (CSA-DP) algorithm in Figure 2, implements value iteration for DCA-MDPs producing a sequence of value functions $V^0, V^1, \ldots$ until convergence and uses the XADD structure to efficiently represent state-action partitions.

We now describe each section of the algorithm using the INVENTORY CONTROLexample.

Algorithm 1: CSA-DP

begin

if R is action independent **then** set $V^0 = R$ **else** $V^0 = 0$

while $h < maxHorizon$ **do**

foreach *action variable* Δ *in A* **do**

Prime Value-function($V^{'h}$)using substitute

foreach *continuous state variable in S* **do**

Perform Continuous integration:

$$\tilde{Q}_{a_\Delta^{h+1}} := \int_{x'_j} \delta[x'_j - g(\vec{x})] V'^h dx'_j \quad (11)$$

$$= V'^h\{x'_j/g(\vec{x})\} \quad (12)$$

foreach *discrete state variable in S* **do**

Perform Discrete marginalization:

$$\tilde{Q}_{a_\Delta^{h+1}} := \left[\tilde{Q}_{a_\Delta^{h+1}} \otimes P(b'_i|\vec{b},\vec{x},a)\right]|_{b'_i=1} \oplus \left[\tilde{Q}_{a_\Delta^{h+1}} \otimes P(b'_i|\vec{b},\vec{x},a)\right]|_{b'_i=0} \quad (13)$$

foreach *case partition in* $\tilde{Q}_{a_\Delta^{h+1}}$ **do**

1- Compute bounds on Δ: (UB, LB, Roots)

2- Evaluate $\tilde{Q}_{a_\Delta^{h+1}}$ using bounds (Substitute and take maximum)

3- Add bound constraints

$()LB \leq Roots \leq UB)$

Compute new Value function:

$$V^{h+1} = \max(Q_{a_1}^{h+1}, \max(\ldots, \max(Q_{a_{p-1}}^{h+1}, Q_{a_p}^{h+1}))) \quad (14)$$

end

1. The symbolic value iteration algorithm works with the XADD structure for efficiency. This means that each of the i-stage-to-go value functions (V^i) is set to be an XADD. In most problems in this domain, the reward function defines action costs as a penalty to avoid the agent of acting more than required. For this reason we define the first value function equal to zero $V^0 = 0$ which is a tree with only one leaf node [0]. If the reward function of the DCA-MDP is independent of the actions, then the first value function can be equal to this reward $V^0 = R$.

2. Next, for $h > 0$ the algorithm iterates over the horizon until a convergence criteria is met or in our case the maximum number of iterations is reached. The next parts of the algorithm is performed for each horizon. For each horizon we take a continuous action Δ and perform the following steps in sequence.

3. The first step of value iteration for horizon h, action Δ is to prime the current state so it becomes the next state. This can be performed by substituting the state variables with their primed versions. Given as an example in section 2 , we set

$$\sigma = \{d_1/d'_1, \ldots, d_n/d'_n, x_1/x'_1, \ldots, x_m/x'_m\}$$

and obtain $V'^h = V^h\sigma$.

4. Next, we need to consider the back-up step of the Bellman equation of (15) restated here:

$$Q_a^{h+1}(\vec{b},\vec{x}) = R_a(\vec{b},\vec{x}) + \gamma \cdot \sum_{\vec{b}'} \int_{\vec{x}'} \left(\prod_{i=1}^{n} P(b'_i|\vec{b},\vec{x},a) \prod_{j=1}^{m} P(x'_j|\vec{b},\vec{b}',\vec{x},a)\right) V^h(\vec{b}',\vec{x}')d\vec{x}' \quad (15)$$

We first evaluate the integral marginalization $\int_{\vec{x}'}$ over the continuous variables in the algorithm. For each of the continuous variables x'_j, the only functions dependent on this variable are V'^h and $P(x'_j|\vec{b},\vec{b}',\vec{x},a) = \delta[x'_j - g(\vec{x})]$; hence, marginal over x'_j need only be computed over their product. Also according to [] this integration is equal to substituting $\sigma = \{x'_j/g(\vec{x})\}$ on V'^h as shown in (11) and (12) in the algorithm.

This substitution is performed for each x'_j ($1 \leq j \leq m$) updating $\tilde{Q}_a^{h+1}$ each time, then after elimination of all x'_j the partial regression of V'^h for the continuous variables for each action a is denoted by $\tilde{Q}_a^{h+1}$. Substitution preserves the disjoint partition property, so given that the previous value function is also a case statement, substituting it with the transition function which is again in the form of a case statement, produces $\tilde{Q}_a^{h+1}$ as a case statement that can also be presented by an XADD.

5. The next step is to complete the backup step using $\tilde{Q}_a^{h+1}$. For this we consider marginalization of discrete variables $\sum_{\vec{d}'}$ in (15). The discrete regression applies the following iterative process for each d'_i according to (13). As explained in an example for the restriction operator $|_v$ the variable v is set to the given value if present. Both Q_a^{h+1} and $P(b'_i|\vec{b},\vec{x},a)$ are represented as case statements and restriction preserves this property, thus after marginalizing over all $\vec{b}'$, the resulting $\tilde{Q_a^{h+1}}$ is the symbolic representation of the intended Q-function.

6. Now that we have Q_a^{h+1} in the case format we perform continuous maximization for action Δ as explained in section (2.1). Figure 3 also explains this step visually using XADDs from the INVENTORY CONTROLproblem. We take each case statement which are disjoint partitions on the state-action space independently and keep a running maximum for the result. In Figure 3, we shown 2 of the ?? branches and consider the steps of our algorithm on them. For each such partition, we first build the bounds on Δ based on the nodes:
   (a) Consider two XADDs for the UB, LB and the root
   (b) For each decision node , isolate the action Δ variable on the LHS and take everything to the RHS.
   (c) Divide the RHS by the coefficent of Δ
   (d) Flip the inequality sign if the false branch of the parent decision is considered or the coefficient of Δ is negative.
   (e) Consider the RHS as the UB if inequality is ($<$, $\leq$), else add RHS to the LB.
   (f) Perform Symbolic Minimum on the upper bound XADD and Symbolic Maximum on the lower bounds
   (g) Take first derivative of the Leaf of this case partition and equal that to zero
   (h) If a value (constant or variable) appears for action Δ this is the root value.(XADD with single node)
   (i) The two resulting XADDs now have the UB, LB and roots for this branch

   Figure 3 shows the XADDs for the UB,LB and root(in this case it is a constant value of 0, but it can take any symbolic function form). Having obtained the XADDs for the bounds, we now evaluate the bounds on the Q-function value at the leaf. This is equal to substituting the bounds in the leaf for Δ:

   We then take the maximum of the bounds with regard to the new XADDs. This is equal to the Symbolic maximum over three different functions which is an XADD itself:

   In the last step, we add the constaints for the bounds as well as the action-independent decisions of this partition to the resulting XADD. Figure 3 shows the end result XADD for both partitions. The running maximum which is an initially empty XADD is now used to take the maximum of each branch after they are produced. By the time all the case statements of $\tilde{Q_a^{h+1}}$ have been processed in this algorithm, the running maximum will contain the maximum over all the partitions which is independent of action Δ now, or the V_Δ^{h+1}

   This algorithm is performed for each of the continuous actions in $a \in \{a_1, \ldots, a_p\}$ separately. Obtaining the overall V^{h+1} in case format as defined in (10) requires sequentially applying *symbolic maximization* as defined in (14).

In the next section we provide practical results for this algorithm in various domains.

6 Empirical Results

6.1 Domains

WATER RESERVOIR The problem of WATER RESERVOIRneeds to make an optimal decision on how much and when to discharge water from multiple water reservoirs to maximize hydroelectric energy productions while considering environment constraints such as irrigation requirements and flood prevention. This problem is often considered using one single reservoir since in the multiple case the outflow of upstream reservoirs affect the inflow of downstream reservoirs. Mathematical multivariate models such as autoregressive processes using the historical mean and variance of a site is used []. The state is the tuple (l, i, e) with the continuous variable $l_i \in \mathbb{R}$ the water level of the i-th reservoir,

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.[] Using our method for continuous action value iteration, we show that this problem can be handled efficiently and is scalable to multi-reservoir problems.

MULTI-RESERVOIR 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 can be considered both continuous or discrete, here we assume the customers either want high energy around 700 units(e.g. in winter or summer) or low energy of about 350 units (e.g. in spring or autumn). The discharge actions of both reservoirs are considered continuous $(d1, d2)$. For the transition we define the following equations:

$$l1' = \begin{cases} i: & l1 + 100 + d2 - d1 \\ \neg i: & l1 + 50 + d2 - d1 \end{cases}$$

$$l2' = \begin{cases} i: & l2 + 100 - d2 \\ \neg i: & l2 + 50 - d2 \end{cases}$$

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

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

As for the reward function, we consider a reward of the energy demand multiplied by the price per unit if the demand is fulfilled. The energy If the demand is not reached, then only the amount discharged will get this reward. There is also a penalty for reaching low water levels and a higher penalty for very high water levels to prevent unsatisfied customers and flooding. The reward function can be formulized as below:

$$R = \begin{cases} (d1 + d2) \geq 700 \wedge (50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*700} \\ (d1 + d2) \geq 700 \wedge \neg(50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*700-0.5*(50-l1-l2)-0.01*(l1+l2)} \\ (d1 + d2) \leq 700 \wedge (50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*(d1+d2)} \\ (d1 + d2) \leq 700 \wedge \neg(50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*(d1+d2)-0.5*(50-l1-l2 -0.01*(l1+l2)} \\ (d1 + d2) \geq 350 \wedge (50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*350} \\ (d1 + d2) \geq 350 \wedge \neg(50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*350-0.5*(50-l1-l2)-0.01*(l1+l2)} \\ (d1 + d2) \leq 350 \wedge (50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*(d1+d2)} \\ (d1 + d2) \leq 350 \wedge \neg(50 \leq 4000 \wedge 50 \leq 4000): \\ \textit{0.3*(d1+d2)-0.5*(50-l1-l2)-0.01*(l1+l2)} \end{cases}$$

According to the transition and reward function, the value iteration is performed for this problem using the algorithm explained in section 4.

Refinement reservoir In order to demonstrate the effectiveness of planning with our model, we use a 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 SDP framework. This means that as the horizon increases, the value function will have better values, while the value for lower horizons were less than the current value function. We want to prevent the upstream reservoir 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 to meet customer electricity demands. The inflow to both reservoirs is assumed to be the same (same amount of rainfall) 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 draining water from one reservoir to the other, or the choice of not draining 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 means that higher elapsed time is desired to maximize the reward, while lower ones ensure flood prevention.

6.2 Results

7 Related Work

8 Conclusions

Acknowledgements