Exact Symbolic Dynamic Programming for Continuous POMDPs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Decision-theoretic planning problems try to find the optimal sequence of actions for a given environment. Using continuous models in such problems is naturally closer to the real-world. Partially Observable MDPs model uncertain environments where the underlying state is not be fully observable.

While previous work have provided solutions for discrete states and/or observations, here we define the first exact solution to both continuous state and observations. This symbolic dynamic programming approach avoids enumerating the state and observation space providing solutions for our general discrete and continuous partially observable MDP. The main contribution of this paper is to use continuous states with discrete observations and extending it to continuous observations by defining relevant observation partitions associated with each state partition. This allows us to perform the Bellman backup operation efficiently. We address the problems in POMDP continuous planning and solutions to solve them symbolically.

1 Introduction

In many complex systems complete information of the state is not available due to noisy or poor sensors and readings of them. Partially observable Markov Decision Processes (POMDPs)[2] can be used for planning under such uncertainty in abstract problems. In many real-world problems, resources and sensor inputs are naturally modeled using continuous variables.

Unfortunately most POMDP techniques can not be applied to such settings without assuming prior knowledge about the problem domain. Most methods use some approximation to discretize the continuous state, observation or action space which increases computational complexity [9]. Exact solutions to a multi-variate continuous state and observation setting has not been approached for the curse of dimensionality problem. In assuming a continuous observation setting, the agent must search for all possible observations that are read from the environment. This can quickly become infeasible even for small problem sizes.

In this paper we provide an exact solution to a subset of POMDPs with continuous states and observations. Our symbolic dynamic programming (SDP) approach assumes discrete noise and piecewise linear dynamics and functions. We formulize our problems using a Discrete and Continuous POMDP (DC-POMDP) model and apply our SDP algorithm. The main problem that rises in any problem modeled using DC-POMDP is the definition of different observation partitions and their probability. We apply symbolic solutions such as integration and maximization to provide an optimal closed-form solution to our problem domains. As an example we consider the following power plant example which has mainly been solved in the literature using discrete sets of states and observations [1]. We use the example throughout the paper to help better understand the definitions.

Example (POWER PLANT) *The steam generation system of a power plant aims to provide hot steam to the steam turbine which in turn produces electricity. Feed-water is exposed to heat in the tubes leading to the water tank . Evaporating the water is performed under specific pressure and temperature inside the attached water tubes. Mixing water and steam pressures makes reading tank levels and temperatures very hard and uncertain. For this reason, the process is modeled using POMDP in the literature. The temperature is assumed to be a continuous variable that is not observed. Noisy observations of the temperature are used to decide on the action of closing or opening the valve.*

We need to extend the basic SDP framework from [6] by adding continuous observations to the model. The solution to a closed-form value function is similar to the algorithm from [4] apart from the fact that no explicit observation is defined in the continuous case. We redefine the value-iteration algorithm using case representations that are implemented using Extended Algebric decision diagrams (XADD) and provide an efficient automatic way to partition the entire observation space into piecewise linear functions where each partition has a relative probability. We use the univariate POWER PLANT example to define the main operations and algorithms and also provide empirical results for a multi-variate version of the problem.

2 DC-POMDP Model

We assume familiarity with MDPs and introduce Discrete and Continuous Partially Observable MDPs as an extension to Discrete and Continuous State MDPs [6]. A Discrete and Continuous partially observable MDP (DC-POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \gamma, h \rangle$. States are represented by vectors of $(\vec{d_s}, \vec{x_s}) = (d_{s_1}, \ldots, d_{s_n}, x_{s_1}, \ldots, x_{s_m})$ where each $d_{s_i} \in \{0, 1\}$ $(1 \leq i \leq n)$ is a discrete boolean and each $x_{s_j} \in \mathbb{R}$ $(1 \leq j \leq m)$ is a continuous variable. Actions are presented by a finite set of actions $\{a_1, \ldots, a_p\}$. We define discrete and continuous observations by the vector $(\vec{d_o}, \vec{x_o}) = (d_{o_1}, \ldots, d_{o_n}, x_{o_1}, \ldots, x_{o_m})$ where each $d_{o_i} \in \{0, 1\}$ $(1 \leq i \leq n)$ is boolean and each $x_{o_j} \in \mathcal{R}$ $(1 \leq j \leq m)$ is a continuous variable.

Three functions are required for modeling DC-POMDPs: (1) $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ a Markovian transition model defined as the probability of the next state given the action and previous state; (2) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function which returns the immediate reward of taking an action in some state and (3) an observation function defined as $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$ which gives the probability of an observation given the outcome of a state after executing an action. A discount factor $\gamma,\ 0 \leq \gamma \leq 1$ is included in the model to account for future rewards in higher horizons.

To define the transition function, we use the fact that the state variables of this POMDP can be factored into a Dynamic Bayesian Network (DBN). Not allowing synchronic arcs between the discrete and continuous variables defines their joint transition to be independent:

$$P(\vec{d_s}', \vec{x_s}' | \vec{d_s}, \vec{x_s}, a) = \prod_{i=1}^{n} P(d'_{s_i} | \vec{d_s}, \vec{x_s}, a) \prod_{j=1}^{m} P(x'_{s_j} | \vec{d_s}, \vec{d_s}', \vec{x_s}, a).$$

The joint observation function can also be defined considering the direct correspondence between the state and observation variables:

$$P(\vec{d_o}, \vec{x_o} | \vec{d_s}, \vec{x_s}, a) = \prod_{i=1}^{n} P(d_{o_i} | d_{s_i}, a) \prod_{j=1}^{m} P(x_{o_j} | x_{s_j}, a).$$

The *binary* variables are represented using conditional probability functions (CPF) in the form of $P(d'_i | \vec{d}, \vec{x}, a)$. For *continuous* variables the CPF $P(x'_j | \vec{d}, \vec{d'}, \vec{x}, a)$ is represented by *piecewise linear equations* (PLEs) that are first-order Markov and deterministic, i.e. the probabilities are encoded using the Dirac $\delta[\cdot]$ function. The reward function $R(\vec{d}, \vec{x}, a)$ is set to be any general piecewise linear function that could facilitate pruning for space efficiency via bilinear solvers.

For the POWER PLANT example, the temperature $t_s \in \mathbb{R}$ is modelled as the continuous state variable. Noisy observation of the temperature $t_o \in \mathbb{R}$ is used to help decide the optimal action to either open or close the pressure valve $a \in \{open, close\}$. For the transition probability, we use the Dirac function to model the deterministic equations. We can also define stochasticity in the transition

model using boolean random variables that are sampled stochastically. The reward function can be any linear function of the state or action. Going above a threshold temperature (e.g. $T = 10$) will cause an explosion in the plant when trying to close the valve which results in the negative reward of -1000. Staying below this temperature is safe and will produce electricity an gain the reward of 100. The reward of an open valve is -1.

$$P(t'_s|\vec{t_s}, a) = \delta\left[t'_s - \begin{cases}(a = open) & : t_s - 5\\ (a \neq open) & : t_s + 7\end{cases}\right], R(t_s, a) = \begin{cases}(a = open) & : -1\\ (a \neq open) \wedge (t_s > T) & : -1000\\ (a \neq open) \wedge \neg(t_s > T) & : 100\end{cases}$$

This example will try to maximize its reward using observations on the temperature. The observations recieved before closing the plant to a safe temperature are therefore critical. For the observation model we begin with a discrete model for two observations t_{o_1}, t_{o_2} such as the following:

$$P(t_{o_1}|t'_s, open) = \begin{cases}\delta\left[t_s <= 15\right] & : 0.9\\ \delta\left[\neg(t_s <= 15)\right] & : 0.1\end{cases}, P(t_{o_2}|t'_s, open) = \begin{cases}\delta\left[t_s <= 15\right] & : 0.1\\ \delta\left[\neg(t_s <= 15)\right] & : 0.9\end{cases}$$

$$P(t_{o_1}|t'_s, close) = \begin{cases}\delta\left[t_s <= 15\right] & : 0.1\\ \delta\left[\neg(t_s <= 15)\right] & : 0.8\end{cases}, P(t_{o_2}|t'_s, close) = \begin{cases}\delta\left[t_s <= 15\right] & : 0.9\\ \delta\left[\neg(t_s <= 15)\right] & : 0.2\end{cases}$$

After defining the solution to the discrete observation with a continuous state DC-POMDP, we can extend this to consider continuous observations using a continuous uniform noise distribution which is more realistic in many domains. For a certain interval, the probability is a fixed number, while it is zero elsewhere. The integral of the uniform distribution should sum to one.

$$P(t_o|t'_s, open) = \begin{cases}(t_o > t_s - 5) \wedge (t_o < t_s + 5) & : 0.1\\ \neg((t_o > t_s - 5) \wedge (t_o < t_s + 5)) & : 0\end{cases}$$

Next we define the general solution to solving DC-POMDPs.

3 DC-POMDP solution

The DC-POMDP will choose actions in order to compute an optimal plan over time. This plan is the policy π of the agent depending on the history of actions and observations over a Markovian state. In the partially observable case where there is uncertainty on the states, the policy is a function over a continuous set of probability distributions over $|\mathcal{S}|$. Observations don't provide a unique state therefore the probability of the state is stored as the belief state b.

At any of the horizon steps, a DC-POMDP policy $\pi(b)$ is defined as the iterative plan on executing an action and observing all probable outcomes of the set of observations from the $(h-1)$-step. The belief state is updated after this according to the Bayes rule: $\sum_{d_s}\int_{x_s} b(xd_s)T(xd'_s, xd_s, a)Z(xd'_s, xd_o, a)$ where for simiplicity we use $(xd_s = x_s, d_s)$ and $(xd_o = x_o, d_o)$. The optimal policy is defined over the belief state as the sum of the expected discounted rewards over horizon h starting with belief b_0.

$$V^h_{\Pi^*}(\vec{b}) = E_{\pi^*}\left[\sum_{h=0}^{H}\gamma^t \cdot r_h \middle| \vec{b}_0 = \vec{b}\right]$$

Solving the value function for the optimal policy is performed using Dynamic programming algorithms such as value iteration. The decision at each time step is to pick an action based on past actions and observations. The possible set of conditional plans β^h correspond to a decision tree. Figure 1 left, shows the decision tree for the simple case of three horizons with discrete observations. It represents a conditional plan consisting of an action and two possible observation strategies. For continuous observations, conditional plans are decision trees with infinitely many branches. In our solution we show that instead of a policy tree with infinite branches, we can create partitions based on relevant observation values, resulting in a policy tree similar to that of figure 1.

We define the value iteration algorithm for discrete observations and discrete states that finds the optimal value for each horizon starting with an initial belief. Although the number of belief states

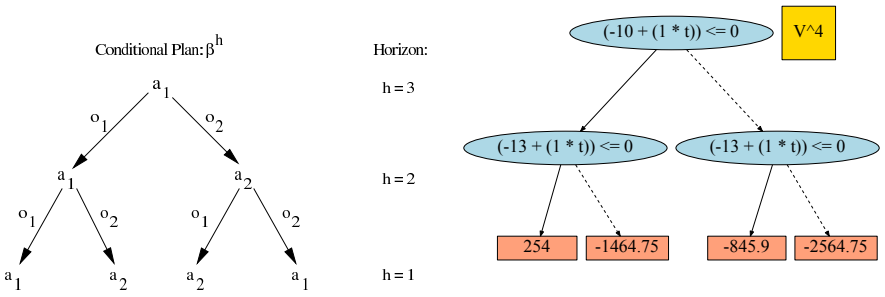

Figure 1: Left: Example conditional plan β^h for discrete observations. Right: The optimal value function for POWER PLANT as a decision diagram: the *true* branch is solid, the *false* branch is dashed.

is infinite, the optimal value function for h is proven to be a piecewise linear and convex function of the belief state [7]. The optimal value function is the maximum value over a finite set of alpha vectors α_i^h:

$$V_{\Pi^*}^h(\vec{b}) = \max_{\alpha_i^h \in \Gamma^h} \alpha_i^h \cdot \vec{b} \tag{1}$$

where the belief states $\vec{b}$ and each $\alpha_i^h \in \Gamma^h$ are of dimension $|\mathcal{S}|$. At each of the h-steps the value function is parameterized by the set of α-vectors that partition the belief state. At a particular belief point, we can compute the next value function:

$$V^{h+1}(b) = \max_a \left(b \cdot r_a + \gamma \sum_o p(xd_o|a,b) V^h(b|xd_o, a) \right)$$

Replacing the previous iteration value with (3) and also updating the belief after executing action a and observing xd_o results in the following equation:

$$V^{h+1}(b) = \max_a \left(b \cdot r_a + \gamma \sum_o \max_{\langle g_{a,o,j}^h \rangle_j} b \cdot g_{a,o,j}^h \right)$$

where $r_a = r(x_s, d_s, a)$ and $g_{a,o,j}^h = \int_{x_{s'}} \sum_{d_{s'}} p(xd_o|xd'_s, a) p(xd'_s|xd_s, a) \alpha_j^h(xd'_s)$

Calculating all possible ways of building the value function independent of a certain belief state can define the optimal value function. To derive the Γ-set of all possible α-vectors for each horizon, we use the value iteration algorithm from [4]. Initializing $\alpha_1^0 = \vec{0}$ and $\Gamma^0 = \{\alpha_1^0\}$, Γ^h is obtained from Γ^{h-1} using the backup operation defined in (4):[1]

$$\Gamma_a^h = r_a + \gamma \boxplus_{o \in \mathcal{O}} \left\{ g_{a,o,j}^h(\cdot) \right\}_j ; \forall \alpha_j^{h-1} \in \Gamma^{h-1} \tag{2}$$

$$\Gamma^h = \bigcup_a \Gamma_a^h \tag{3}$$

Each action-dependent Γ sums over the possible set of $g_{a,o,j}^h(\cdot)$ given an observation model to derive discrete observations xd_o and for all $\alpha_j^{h-1} \in \Gamma^{h-1}$. The value $g_{a,o,j}^h(\cdot)$ is computed by summing over all states given the probability functions and previous α-vectors. For each horizon, the optimal α-vector is obtained from the maximum over all α-vectors: $\alpha^* = \arg\max_{\alpha_i^h \in \Gamma^h} \alpha_i^h \cdot b$

Here the cross-sum operation takes the set of α -vectors for each observation and produces a sum over vectors that depends on the number of previous iteration vectors. Thus the α-vector set grows exponentially in the number of observations, i.e., $|\Gamma^h| = |\mathcal{A}||\Gamma^{h-1}|^{|\mathcal{O}|}$ during the cross-sum operation. For this reason even for a small number of discrete observations, tractable solutions may be hard to define. Here we do not restrict our observations to discrete ones, but extend it to a fully continuous approach where continuous observations and states are allowed in the DC-POMDP. We use the number of observation partitions $|\mathcal{O}|$ as the branching factor in the policy tree. We present the symbolic value iteration algorithm in the case of continuous observations using relevant observation partitioning of this continuous space.

[1]The $\boxplus$ of sets is defined as $\boxplus_{j \in \{1,\ldots,n\}} S_j = S_1 \boxplus \cdots \boxplus S_n$ where the pairwise cross-sum $P \boxplus Q = \{\vec{p} + \vec{q} | \vec{p} \in P, \vec{q} \in Q\}$.

4 Symbolic Dynamic Programming

Dynamic programming provides exact solutions to POMDP problems for small domains. As the domain grows, fully enumerating the state, action and observation space leads to intractable solutions even for low horizons. Avoiding this enumeration often requires approximating the solutions. We claim an exact solution using symbolic dynamic programming. The SDP solution for fully observable domains has been presented in [6] Our symbolic algorithm deals with continuous observations and states by generating the relevant observation partition at every time step and performing the backup operation using α-vectors. We define the requirements to implement the value iteration algorithm for DC-POMDP.

4.1 Case Representation and Extended ADDs

The POWER PLANT example represented all functions using the case structure which can be generally defined as:

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

Where ϕ_i are disjoint logical formulae defined over the state $(\vec{d}, \vec{x})$ with logical $(\wedge, \vee, \neg)$ combinations of boolean variables and inequalities $(\geq, >, \leq, <)$ over continuous variables. The f_i are function definitions either linear or quadratic in the continuous parameters.

For *unary operations* such as scalar multiplication $c \cdot f$ (for some constant $c \in \mathbb{R}$) or negation $-f$ on case statements is simply to apply the operation on each case partition f_i $(1 \leq i \leq k)$.

A *binary operation* on two case statements, takes the cross-product of the logical partitions of each case statement and performs the corresponding operation on the resulting paired partitions. The cross-sum $\oplus$ of two cases is defined as the following:

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

Likewise $\ominus$ and $\otimes$ are defined by subtracting or multiplying partition values. Inconsistent partitions can be discarded when they are irrelevant to the function value. The next three operations are required in defining the value iteration algorithm. A *symbolic case maximization* is defined as below:

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

Restriction takes a function f to restrict 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:

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

Symbolic substitution simply takes a set σ of variables and their substitutions, where the LHS of the substitution operator $/$ represents the substitution variable and the RHS of the $/$ represents the

Algorithm 1: VI(DC-POMDP, H,$ContObs$, b_i) $\longrightarrow$ $(V^h, \pi^{*,h})$

begin
  $V^0 := 0, h := 0$
  while $h < H$ **do**
    $h := h + 1, \Gamma^h := \emptyset$
    foreach $a \in A$ **do**
      $\Gamma_a^h := \emptyset$
      if $ContObs = true$ **then**
        $P(xd_{o_i}|xd_{s_i}) :=$ `GenRelObs` (Γ^{h-1}, a, b_i)
      foreach $xd_{o_i} \in \mathrm{O}^{h-1}$ **do**
        foreach $\alpha_j^{h-1} \in \Gamma^{h-1}$ **do**
          $\alpha_j^{h'} =$ `Prime` (α_j^h)
          // $\forall\ d_i \rightarrow d_i'$ and $\forall x_i \rightarrow x_i'$
          $\Gamma^h_{a,xd_{o_i},j} := P(xd_{o_i}|xd_{s_i}) \otimes$ `Backup` $(\alpha_j^{h'}, a)$
        $\Gamma^h_{a,xd_o} := \boxplus \Gamma^h_{a,xd_{o_i}}$
      $\Gamma_a^h := R_a \oplus \gamma \cdot \Gamma^h_{a,xd_o}$
      $\Gamma^h := \Gamma^h \cup \Gamma_a^h$
    $\Gamma^h :=$ `Prune` (Γ^h)
    $V^h := \max_{\alpha_j \in \Gamma^h} b_i \cdot \alpha_j$
    $\pi^{*,h} := \arg\max_a \Gamma_a^h$
    if $V^h = V^{h-1}$ **then**
      break *// Terminate if early convergence*
  return $(V^h, \pi^{*,h})$
end

expression that should be substituted in its place. Hence to perform a continuous regression on a more general representation, we obtain that $\int_{x_j'} P(x_j'|\cdots)V'^h dx_j'$

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

The data structure of *extended ADD* (XADD) [6] is used to support case statements and the required operations. Here we use continuous SDP for both states and observations in DC-POMDPs. The value functions and α-vectors partition into states of equal value, both of which can be represented using XADDs. For increasing efficiency linear constraint feasibility checkers such as LP solvers allow pruning unreachable paths in the XADD. The right picture in Figure 1 is an example of an XADD representation.

4.2 Value iteration for DC-POMDPs

For continuous state and discrete observations, we use the symbolic version of Monahan's algorithm as the DC-POMDP solution. The same approach is required for continuous observations only based on point-based dynamic programming backups[]. Here for a given belief, the best conditional plan corresponding to an α-vector is computed using point-based backups:

$$\alpha^{h+1}(b) = R(b) + \gamma \int_{xd_o} P(xd_o|a,b)\alpha^{h+1}(b)$$

where $P(xd_o|a,b) = \int_{xd_s} \int_{xd_s'} b(s)P(xd_s'|xd_s,a)P(xd_o|xd_s',a)$

Given an initial set of beliefs, we can compute the relevant observation partitions explained in the next section and then continue in a similar fashion of having discrete observations. To visualize

we explain of the value iteration algorithm as defined in VI for DC-POMDPs. Each step of the algorithm is explained using the POWER PLANT example for the first three horizons. Note that if the input to this algorithm is continuous observations $ContObs = true$ then line 8 finds the relevant observation partitions, else discrete observations are assumed and their probabilities are given in the problem specifications. Also a belief is given as the input and often several beliefs are required to cover the belief space and thus the algorithm is executed for each one of $b_i \in \mathbb{B}$.

For $h = 0$, V^0 is given in line 2. For $h = 1$ we choose an action, for example $close$ to start with. The α-vector set from $h = 0$ is empty so line 7–8 of generating the relevant observation partitions is empty. Also priming the vectors and obtaining the product of relevant observations and the backup is an empty set. In line 15 the reward $R(s, close)$ is added to obtain $\alpha^1_{1,close}$ as defined by (8). The same operation is repeated for the $open$ action where $R(s, open) = -1$. Line 16 defines the final Γ-set: $\Gamma^1 = \{\alpha^1_{1,open}, \alpha^1_{1,close}\}$.

The Backup algorithm is defined in a separate algorithm on the next page which takes the primed version of the variables and performs regression on the α-vectors. *Boolean restriction* is defined in lines 5–8 where $f|_{b=v}$ restricts a boolean variable b occurring in f to values $v \in \{0, 1\}$. This can be done by simply appending $b = v$ to each case partition (logical $\wedge$ of $b = v$ and the logical first-order formulas of f).

Continuous regression $\int f(x'_{s_j}) \otimes P(x'_{s_j}|\cdots)dx'_{s_j}$ in line 3–4 has the form $\delta[x'_{s_j} - h(\vec{z})]$ for the probability over the next state, where $h(\vec{z})$ is a case statement and $\vec{z}$ does not contain x'_{s_j}. A symbolic substitution is defined for this integration such that any occurrence of x'_{s_j} in $f(x'_{s_j})$ is substituted with $h(\vec{z})$: $\int f(x'_{s_j}) \otimes \delta[x'_{s_j} - h(\vec{z})]dx'_{s_j} = f(x'_j)\{x'_{s_j}/h(\vec{z})\}$

For $h = 2$, for each of the discrete actions, first we generate the probability of the relevant set of observations in the GenRelObs algorithm. This has been done for the $close$ action in the next section and we use the results here.

$$P(z_k) = \begin{cases} (2 < t_o < 6) & : 0.2 \\ (-4 < t_o < 2) & : 0.6 \\ (-8 < t_o < -4) & : 0.1999 \end{cases}$$

Next for each of the relevant partitions and both α-vectors of the previous Γ set lines 9–16 are executed. Line 11 will substitute prime values for all state variables. Line 13 produces the backup α-vector and multiplies it by the weight derived from the relevant observation partition. For each observation, two vectors are formed for each corresponding α-vector. For example line 13 for the observation partitions above and α^1_1, α^1_2 is defined as:

$$\Gamma^2_{close,z_1,1} = \begin{cases} (t'_s > 15) & : -600 \\ \neg(t'_s > 15) & : 60 \end{cases} \qquad \Gamma^2_{close,z_1,2} = \top : -0.6$$

$$\Gamma^2_{close,z_2,1} = \begin{cases} (t'_s > 15) & : -199.999 \\ \neg(t'_s > 15) & : 19.999 \end{cases} \qquad \Gamma^2_{close,z_2,2} = \top : -0.1999$$

$$\Gamma^2_{close,z_3,1} = \begin{cases} (t'_s > 15) & : -200 \\ \neg(t'_s > 15) & : 20 \end{cases} \qquad \Gamma^2_{close,z_3,2} = \top : -0.2$$

Next in line 14 we have to take the cross-sum of the two sets which takes every possible combination of observations according to the α-vectors.

$$\Gamma^2_{close,1} = \begin{cases} (t'_s > 15) & : -800.2 \\ \neg(t'_s > 15) & : 79.8 \end{cases} \qquad \Gamma^2_{close,2} = \begin{cases} (t'_s > 15) & : -1000 \\ \neg(t'_s > 15) & : 100 \end{cases}$$

$$\Gamma^2_{close,3} = \begin{cases} (t'_s > 15) & : -800.2 \\ \neg(t'_s > 15) & : 79.8 \end{cases} \qquad \Gamma^2_{close,4} = \begin{cases} (t'_s > 15) & : -600.4 \\ \neg(t'_s > 15) & : 59.6 \end{cases}$$

$$\Gamma^2_{close,5} = \begin{cases} (t'_s > 15) & : -400.5999 \\ \neg(t'_s > 15) & : 39.3999 \end{cases} \qquad \Gamma^2_{close,6} = \begin{cases} (t'_s > 15) & : -200.7999 \\ \neg(t'_s > 15) & : 19.999 \end{cases}$$

$$\Gamma^2_{close,7} = \begin{cases} (t'_s > 15) & : -200.8 \\ \neg(t'_s > 15) & : 19.2 \end{cases} \qquad \Gamma^2_{close,8} = \top : -1$$

Algorithm 2: `Backup`$(\alpha^{'h}_{j,o}, a, P(z)) \longrightarrow G^h_a$

begin
  // Continuous regression, marginal integration
  for *all* xd'_{s_j} *in* $\alpha^{'h}_{j,o}$ **do**
    $G^h_a := \int P(z) \otimes P(x'_{s_j}|\vec{d}_s, \vec{d}_s{}', \vec{x_s}, a)\, d_{x'_{s_j}}$
  // Discrete regression, marginal summation
  for *all* d'_{s_i} *in* $\alpha^{'h}_{j,o}$ **do**
    $G^h_a := \left[G^h_a \otimes P(d'_{s_i}|\vec{d}_s, \vec{x_s}, a)\right] |_{d'_{s_i}=1}$
      $\oplus \left[G^h_a \otimes P(d'_{s_i}|\vec{d}_s, \vec{x_s}, a)\right] |_{d'_{s_i}=0}$
  return G^h_a
end

Algorithm 3: `Prune`$(\Gamma^h) \longrightarrow (V^h)$

begin
  $V^h := 0$;
  foreach $\alpha^h_j \in \Gamma^h$ **do**
    $\alpha^h_j :=$ `LP-Solve(`α^h_j`)` ;
    foreach $\alpha^h_k \in \Gamma^h$ **do**
      if $j \neq k$ **then**
        $V^h := \oplus$ `Dominance(`α^h_j, α^h_k`)` ;
  return (V^h) ;
end

All 8 vectors are multiplied by the discount γ and added with the reward in line 15 where the addition is an augmented $\oplus$ for adding each vector with the reward function. The same operation is performed for the *open* action and then in line 15 the Γ set for each action are added to the set of Γ^h to build all the vectors for the next horizon.

In line 17 this final set is using in the `Prune` algorithm. For all α-vectors in the current Γ-set first an LP-solver is used to prune any inconsistency in the branches of this vector. Next a pairwise dominance test is performed where for all j, k: $\alpha_j(\vec{s}) \geq \alpha_k(\vec{s})$ The result of this pairwise test is a dominant α-vector which is added to the set of Γ^h. In the last step, we find the α-vector that maximizes the value-function with respect to this belief and save it as the new Γ-set for the next horizon. The action from which this α-vector was chosen from is the maximum policy. This operation is performed for each of the belief points $b_i \in B$ and for each belief point one dominant α-vector is stored in the Γ-set. The algorithm iterates until convergence. In the next section we discuss the main part of our algorithm which is generating relevant observation partitions.

4.3 Continuous Observation Space

The main operations required for value iteration in DC-POMDP is generating relevant observations to perform the bellman backup operation.

The `GenRelObs` algorithm defines the operation of generating relevant observation partitions. We use the second iteration of our `Power Plant` example to demonstrate the steps required to partition the entire continuous observation space. This continuous nature requires assuming some belief state and obtaining the relevant partitions according to that belief. Starting with an empty set of relevant observation partitions z, the algorithm first eliminates the next state variable xd'_s. It multiplies the observation model and the transition model in the set of α-vectors of the previous horizon. $\alpha_j(xd_s, xd_o) := \int_{s'} P(xd_{o_i}|xd'_{s_i}, a) * P(xd'_{s_i}|xd_{s_i}, a) * \alpha_j$ This operation is similar to that of

`Regress` plus the observation regressed model. For continuous variables the continuous regression and for discrete variables boolean restriction is used as defined in the previous section. Assume that after $h = 1$ the set of α-vectors are defined as:

$$\alpha_1^1(s) = \begin{cases} (t_s > 200) & : -1000 \\ \neg(t_s > 200) & : 100 \end{cases} \qquad \alpha_2^1(s) = \top : -1$$

For $h = 2$ the resulting α-vectors after this regression step are functions of the observation and current state:

$$\alpha_1^2(s,o) = \begin{cases} (t_s < 15) \wedge (t_s - 10 < t_o < t_s) & : 10 \\ (t_s > 15) \wedge (t_s - 10 < t_o < t_s) & : -100 \; \alpha_2^2(s,o) \\ \neg(t_s - 10 < t_o < t_s) & : 0 \end{cases} = \begin{cases} (t_s - 10 < t_o < t_s) & : -0.1 \\ \neg(t_s - 10 < t_o < t_s) & : 0 \end{cases}$$

We need the α-vectors to partition the observation space, thus next we eliminate the current state variable. During this integration the current state variable is factored out by multiplying the belief in each of the current α-vectors. The resulting δ-functions only depend on the observation: $\delta_j(xd_o) := \int_{xd_{s_i}} b_i * \alpha_j(xd_s, xd_o)$. For the belief $b_1 : U[t_s; 2, 6]$ the two vectors above the resulting δ-functions are defined below:

$$\delta_1(o) = \begin{cases} (2 < t_o < 6) & : 15 - 2.5 * t_o \\ (-4 < t_o < 2) & : 10 \\ (-8 < t_o < -4) & : 20 + 2.5 * t_o \end{cases} \qquad \delta_2(o) = \begin{cases} (2 < t_o < 6) & : -15 + 0.025 * t_o \\ (-4 < t_o < 2) & : -0.1 \\ (-8 < t_o < -4) & : -0.2 - 0.025 * t_o \end{cases}$$

The observation dependent δ-functions divide the observation space into regions which can yield the optimal policy according to the belief state b_1. According to continuous observation space defined in [3], for a 1-dimensional observation space with two or more functions of the observation, we need to find the optimal boundaries or partitions of the space. They propose numerical solutions to finding the roots of any two observation dependant function. Here we use the symbolic power of the max-operator to find all the maximum boundary regions of two or more δ-functions at the same time. For the two δ-functions above the following partitions of the observation space is derived after taking their maximuum:

$$\max\Big(\delta_1(o), \delta_1(o)\Big) = \begin{cases} (2 < t_o < 6) & : 15 - 2.5 * t_o \\ (-4 < t_o < 2) & : 10 \\ (-8 < t_o < -4) & : 20 + 2.5 * t_o \end{cases}$$

The three partitions define 3 observation regions which we can now use in a similar fashion to a discrete observation set . Thus we only need to find the probability of each of the three distinct observation partitions to use in Monahan's algorithm. If we integrate out the state and observation from the observation model and the belief state regardless of a particular α-vector, it integrates to one: $\int_{xd_o} \int_{xd_s} P(xd_o|xd'_s, a) * P(xd'_s|xd_s, a) * P(s) = 1$

Now that we have partitions on the observation space, we only need to multiply the indicators of each case partition in this formula to obtain the probability of each partition such that it sums to one: $P(z_k) := \int_{xd_o} \int_{xd_s} P(xd_o|xd'_s, a) * P(xd'_s|xd_s, a) * b_i * \mathbb{I}[\phi_{z_k}] d_{xd_o} d_{xd_s}$ For our example the following holds:

$$P(z_k) = \begin{cases} (2 < t_o < 6) & : 0.2 \\ (-4 < t_o < 2) & : 0.6 \\ (-8 < t_o < -4) & : 0.1999 \end{cases}$$

Hence we can now use the algorithms and methods of a discrete observation setting using the probabilities of the partitioned observation space. Next we present some results for 2-Dimensional continuous observation spaces.

5 Empirical Results

We evaluated our continuous POMDP solution using XADDs on the one-dimensional POWER PLANT example and another variant of this problem with two variables. [2] We start by defining the 2-dimensional description of this problem:

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

Algorithm 4: $\texttt{GenRelObs}(\Gamma^h, a, b_i) \longrightarrow P(z)$

begin
  $z := 0$; **for** *all* α_j *in* Γ^{h-1} **do**
    $\alpha_j(xd_s, xd_o) := \int_{s'} P(xd_{o_i}|xd'_{s_i}, a) * P(xd'_{s_i}|xd_{s_i}, a) * \alpha_j$;
  for *all* j *in* $\alpha_j(xd_s, xd_o)$ **do**
    $\delta_j(xd_o) := \int_{xd_{s_i}} b_i * \alpha_j(xd_s, xd_o)\}$
  for *all* j *in* $\delta_j(xd_o)$ **do**
    $z := max(\delta_j(xd_o), z)$
  // ϕ_{z_k} *logical term on each partition of* z
  $P(z_k) := \int_{xd_o} \int_{xd_s} P(xd_o|xd'_s, a) * P(xd'_s|xd_s, a) * b_i * \mathbb{I}[\phi_{z_k}] d_{xd_o} d_{xd_s}$
  return $P(z)$
end

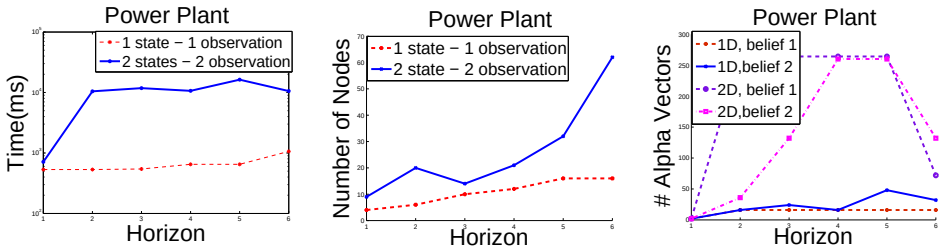

Figure 2: *(left)* Space vs Horizon; *(middle)* Time vs Horizon; *(right)* Number of α-vectors vs Horizon.

2D-POWER PLANT: We consider the more complex model of the power plant similar to [1] where the pressure inside the water tank must be controlled to avoid water mixing into the steam or explosion of the tank. We model the pressure variable p as a partially observable variable from the observation readings of the pressure valve po. The two actions of increase and decrease are defined based on the change in both the temperature and the pressure. For the increase action the transition functions and the reward function is defined as below:

$$P(p'_s|\vec{p_s}, a) = \delta\left[p'_s - \begin{cases}(p+10>20) & :20\\ \neq (p+10>20) & :p_s+10\end{cases}\right]$$

$$P(t'_s|\vec{t_s}, a) = \delta\left[t'_s - t_s + 10\right]$$

$$R(t_s, p_s, a) = \begin{cases}(5\le p_s\le 15)\wedge(95\le t_s\le 105) & :50\\ (5\le p_s\le 15)\wedge(t_s\le 95) & :-1\\ (5\le p_s\le 15)\wedge(t_s\ge 95) & :-3\\ (p_s\le 5) & :-3\\ (p_s\ge 15) & :-5\end{cases}$$

There is a high reward for staying withing the safe temperature and pressure range since it produces power, else depending on how safe it is to have values higher or lower than the safe range, penalty is defined. As for the decrease action, the transition functions reduce the temperature by 5 units and the pressure by 10 units as long as the pressure stays above zero. For the reward function, we assume that there is always a small penalty for decreasing the values because power can not be generated.

For the observation model we consider two continuous uniform distributions such as the following:

$$P(t_o|t'_s) = \begin{cases}(t_o>t_s+80)\wedge(t_o<t_s+105) & :0.4\\ \neg((t_o>t_s+80)\wedge(t_o<t_s+105)) & :0\end{cases}$$

$$P(p_o|p'_s) = \begin{cases}(p_o>p_s)\wedge(p_o<p_s+10) & :0.1\\ \neg((p_o>p_s)\wedge(p_o<p_s+10)) & :0\end{cases}$$

We define two rectangular uniform beliefs around the regions of rewarding, so that one needs to increase the values while the other should decrease them:

$$b_1 = \begin{cases}(p_o>p_s)\wedge(p_o<p_s+10)\wedge(t_o>t_s+90)\wedge(t_o<t_s+100) & :0.01\\ \neg((p_o>p_s)\wedge(p_o<p_s+10)\wedge(t_o>t_s+90)\wedge(t_o<t_s+100)) & :0\end{cases}$$

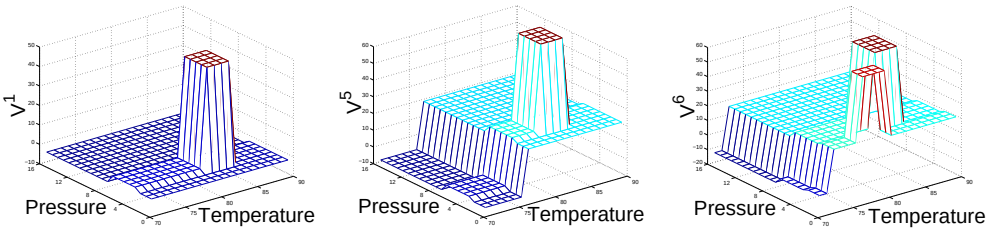

Figure 3: *(left)* Value function for first iteration; *(middle)* $V^5(p,t)$; *(right)* $V^6(p,t)$.

$$b_2 = \begin{cases} (p_o > p_s + 10) \wedge (p_o < p_s + 30) \wedge (t_o > t_s + 90) \wedge (t_o < t_s + 130) & : 0.00125 \\ \neg((p_o > p_s + 10) \wedge (p_o < p_s + 30) \wedge (t_o > t_s + 90) \wedge (t_o < t_s + 130)) & : 0 \end{cases}$$

In Figure 2, a time and space analysis of the two versions of the problem have been performed for up to 6 horizons. Comparing the two problem sizes demonstrates the effect of the number of state-observation variables in our algorithm. As the algorithm progresses, the time required to compute the probability of the partitions and finding the maximum value function increases for both problem sizes and significantly more for the 2D version. The stability in time is due to the fact that each stage almost takes the same time since it has converged quickly.Increase in the problem size, increases the partition numbers on the observation space and this produces more α-vectors which also effecting the space required to perform the algorithm. The number of vectors stays the same for most horizons and they drop after convergence in the 2D problem instance. The number of nodes in each iteration increases as more space is required to perform higher iterations.

In Figure 3 we show plots of the value function for different iterations of the 2D problem instance. Each plot demonstrates the value as a function of the pressure and temperature. Starting with the first iteration, the value is highest for the reward range ($5 < p < 15 \wedge 95 < t < 105$) and -1 or less for other places. In the fifth iteration, the value function has partitioned into more pieces showing how higher temperatures can increase the value without considering the effect of the pressure. In the last plot, horizon 6 has better tuned the value function so that higher temperatures and pressures increase the value function and also within the reward range, finer grain partitions have been formed.

6 Related Work

This work has used the concept of continuous states and the SDP solution to MDPs from [6] and continuous observations from the work of [3]. It is exclusive in the sense that it brings together the continuous state and observation POMDPs using a symbolic approach. In many applications of the POMDP framework, the state and observation space have been considered as discrete values such as [1] here we avoid this general simplification and work with the real values from sensors.

There has been prior work on approximate solutions to POMDPs with large or continuous state spaces [9], [8] and some has been extended to the continuous observation setting using methods such as sampling [5]. In most approximate continuous state POMDP work, continuous or large discrete actions has been used. Thus we can extend the current work using [**?**] to contain continuous actions and provide exact or approximate solutions.

7 Conclusion

We presented the first exact solution to continuous states and observations in a DC-POMDP framework. Transitions and reward functions are defined as piecewise linear functions of the continuous states. Continuous observations require definition of explicit partitions based on the observation model and the belief state of the agent. The key contribution is to define the partitions on the observation space and derive the related probabilities using symbolic integration and maximization. We extend the application to more than one state-observation setting and present the results.

References

[1] Mario Agueda and Pablo Ibarguengoytia. An architecture for planning in uncertain domains. In *Proceedings of the ICTAI 2002 Conference*, Dallas,Texas, 2002.

[2] K.J. Astrom. Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174205, 1965.

[3] Jesse Hoey and Pascal Poupart. Solving pomdps with continuous or large discrete observation spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland, 2005.

[4] G. E. Monahan. Survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[5] J. M. Porta, N. Vlassis, M.T.J. Spaan, and P. Poupart. Point-based value iteration for continuous pomdps. *Journal of Machine Learning Research*, 7:195220, 2006.

[6] Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes de Barros. Symbolic dynamic programming for discrete and continuous state mdps. In *Proceedings of the 27th Conference on Uncertainty in AI (UAI-2011)*, Barcelona, 2011.

[7] R.D. Smallwood and E.J. Sondik. The optimal control of partially observable markov decision processes over a finite horizon. *Operations Research*, 21(5):107188, 1973.

[8] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Articial Intelligence Research (JAIR)*, page 195220, 2005.

[9] S. Thrun. Monte carlo POMDPs. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1064–1070. MIT Press, 2000.