Exact Symbolic Dynamic Programming for Continuous POMDPs

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 defines the 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

The following example is used 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.* Most work in the literature have assumed discretization of the state and observation space [**?**]. Here we take a continuous approach using the DC-POMDP framework.

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 [**?**]. 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) $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ a Markovian transition model defined as the probability of the next state given the action and previous state; (2) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ a reward function which returns the immediate reward of taking an action in some state and (3) an observation function defined as $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0, 1]$ 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.

$$P(t'_s|\vec{t_s}, a) = \delta \left[t'_s - \begin{cases} (a = open) & : t_s + 50 \\ (a \neq open) & : t_s - 35 \end{cases} \right]$$

For observations we can consider discrete noise, modeled using CSEs. The probability that the observed temperature is equal to the actual temperature is 0.9 while it indicates a higher temperature with probability of 0.1. For the open action, observations are more disposed to noise:

$$P(t_o|t'_s, close) = \begin{cases} \delta[t_o = t_s] & : 0.9 \\ \delta[t_o = 1.5 * t_s] & : 0.1 \end{cases} \quad (1)$$

$$P(t_o|t'_s, open) = \begin{cases} \delta[t_o = 1.1 * t_s] & : 0.8 \\ \delta[t_o = 1.4 * t_s] & : 0.2 \end{cases} \quad (2)$$

The reward function can be any linear function of the state or action. Going above a threshold temperature (e.g. $T = 200$) 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.

$$R(t_s, a) = \begin{cases} (a = open) & : -1 \\ (a \neq open) \wedge (t_s > T) & : -1000 \\ (a \neq open) \wedge \neg(t_s > T) & : 100 \end{cases}$$

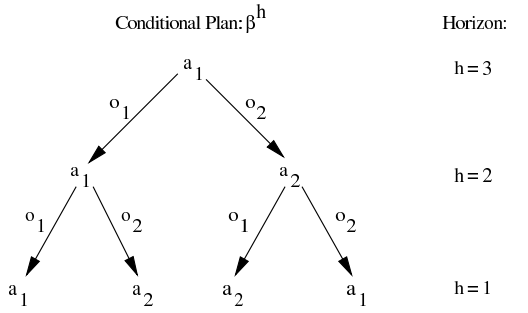

Figure 1: Example conditional plan β^h for discrete observations.

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.

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 which depends 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 a 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.

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

The next section provides the mathematics required to solve this equation in DC-POMDPs.

3.1 Value Iteration solution

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 shows the decision tree for the simple case of three horizons with discrete observations. It represents a condi-

tional 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 prove that a finer grain partitioning of the observation space has the same properties of the minimal partition set. From here on we assume this finite set of observation partitioning as $\mathcal{O}$.

The value iteration algorithm finds the optimal value for each horizon starting with an initial belief. Although the number of belief states is infinite, the optimal value function for h is proven to be a piecewise linear and convex function of the belief state [**?**]. The optimal value function is the maximum value over a finite set of alpha vectors α_i^h:

$$V_{\Pi^*}^h(\vec{b}) = \max_{\alpha_i^h \in \Gamma^h} \alpha_i^h \cdot \vec{b} \tag{3}$$

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

$$g_{a,o,j}^h = \int_{x_{s'}} \sum_{d_{s'}} p(xd_o|xd'_s, a) p(xd'_s|xd_s, a) \alpha_j^h(xd'_s) \tag{4}$$

Calculating all possible ways of building the value function independent of a certain belief state can also 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 (Monahan) [**?**]. 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]

$$\Gamma_a^h = r_a + \gamma \boxplus_{o \in \mathcal{O}} \left\{ g_{a,o,j}^h(\cdot) \right\}_j ; \forall \alpha_j^{h-1} \in \Gamma^{h-1} \tag{5}$$

$$\Gamma^h = \bigcup_a \Gamma_a^h \tag{6}$$

[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 $P \boxplus Q = \{\vec{p} + \vec{q} | \vec{p} \in P, \vec{q} \in Q\}$.

Each action-dependent Γ sums over the possible set of $g_{a,o,j}^h(\cdot)$ given a relevant observation partition (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:

$$\alpha^* = \arg\max_{\alpha_i^h \in \Gamma^h} \alpha_i^h \cdot b$$

For discrete observations, 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 and approaches to solving this problem in the following section.

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 [**?**] 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 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.

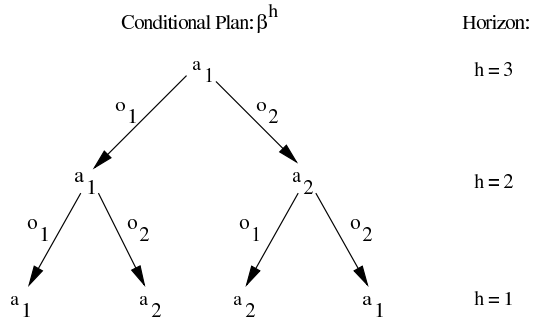

Figure 2: The optimal value function for POWER PLANT as a decision diagram: the *true* branch is solid, the *false* branch is dashed.

Algorithm 1: GenRelObs(Γ^h, a) $\longrightarrow$ O^h

```
1  begin
2      O := ∅
3      for all α'^h_j in Γ^h do
4          Relevant state partitions φ'^h_j of α'^h_j
5          for all xd'_{s_i} in φ'^h_j do
6              φ_{xd_s} := φ{xd_{s_i} \ xd_{o_i}}
7              α^h_{j,o_i} := P(xd_{o_i} | xd'_{s_i}, a) * φ_{xd_s};
8          O^h := ⊕ α'^h_{j,o}
9      return O^h
10 end
```

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 data structure of *extended ADD* (XADD) [**?**] 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.

Algorithm 2: Backup($\alpha'^h_{j,o}, a,$) $\longrightarrow G^h_a$

```
1  begin
2      // Continuous regression, marginal integration
3      for all x'_{s_j} in α'^h_{j,o} do
4          G^h_a := ∫ O^h ⊗ P(x'_{s_j} | d_s, d_s', x_s, a) d_{x'_{s_j}}
5      // Discrete regression, marginal summation
6      for all d'_{s_i} in α'^h_{j,o} do
7          G^h_a := [G^h_a ⊗ P(d'_{s_i} | d_s, x_s, a)] |_{d'_{s_i}=1}
8                   ⊕ [G^h_a ⊗ P(d'_{s_i} | d_s, x_s, a)] |_{d'_{s_i}=0}
9      return G^h_a
10 end
```

4.2 Continuous Observation Space

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

The GenerateRelObs algorithm defines the operation of generating relevant observation partitions. Starting with an empty set of relevant observation partitions O^h, the algorithm iterates through all possible α-vectors of the previous horizon. For each of the vectors, we use the state partitions of that α-vector where the logical formulas ϕ'^h_j of the cases are defined as the state partitions.

Lines 5–7 iterate on all the state variables (both discrete and continuous) of a certain state partition ϕ'^h_j. In line 6 the state variables are substituted with the observation variables based on the observation model provided. In the next line the probability attached to the observation given this state and the input action is multiplied in the relevant observation partition to produce a single case partition with a logical formula depending on observations and a probability based on the observation model. In other words, lines 5-7 take a state partition and return an observation partition that is only relevant to the states according to $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0, 1]$. Finally for every specific α-vector, the relevant observation partitions are added to the set of all relevant observations O^h regardless of the vector they came from. The operator $\oplus$ only performs the operation on the logical formulas ϕ_i of the case partitions,without actually adding the function values.

To illustrate the algorithm with an example, we assume a discrete noise distribution model such as (1). We explain each step of the algorithm using the Power Plant example. Assume that after $h = 1$ the set of α-vectors are defined as:

$$\alpha^1_1(s) = \begin{cases} (t_s > 200) & : -1000 \\ \neg(t_s > 200) & : 100 \end{cases} \alpha^1_2(s) \quad = \top : -1 \tag{7}$$

The relevant state partition only considers the logical formulas: $\{t_s > 200, t_s < 200\}$. For horizon $h = 2$ and specific action $close$ we derive the relevant set of observations for both α_1^1 and α_2^1. The discrete noise observation model (1) is used for the substitution step where t_o is substituted with t_s most of the times(i.e, with probability 0.9) and $2/3 t_o = t_s$ with probability of 0.1. By replacing these inside the state partitions, the relevant observation partitions are formed:

$$\alpha_1^1(t'_s > 200) = \begin{cases} t_o > 200 \\ t_o > 300 \end{cases}$$

$$\alpha_1^1(t'_s < 200) = \begin{cases} t_o < 200 \\ t_o < 300 \end{cases} \qquad \alpha_2^1(t'_s > 200) = \top$$

The probabilities of each partition is then multiplied according to the noise model and the two vectors are added to the relevant observation set O^h. In our example α_2^1 will leave no effect in the final set proposing a partitioning on the observation space depending on the relevant state space partition. The probability of receiving observations between $(200, 300)$,given that the next state has high temperatures (higher than 200) is 0.9. We name the three partitions as the observation partitions of O_1, O_2, O_3:

$$O_1 : P(200 < t_o < 300 | t'_s, close) = \begin{cases} t'_s > 200 & : 0.9 \\ \neg(t'_s > 200) & : 0.1 \end{cases}$$

$$O_2 : P(t_o > 300 | t'_s, close) = \begin{cases} t'_s > 200 & : 0.1 \\ \neg(t'_s > 200) & : 0.0 \end{cases}$$

$$O_3 : P(t_o < 200 | t'_s, close) = \begin{cases} t'_s < 200 & : 0.9 \\ \neg(t'_s > 200) & : 0.0 \end{cases}$$

Backup of an observation-weighted transition function where the weights are determined according to the relevant observation partitions is explained in the `Regress`. It is composed of two main parts:

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 $h(\vec{z})$: $\int f(x'_{s_j}) \otimes \delta[x'_{s_j} - h(\vec{z})] dx'_{s_j} = f(x'_j)\{x'_{s_j}/h(\vec{z})\}$

In our example the α-vectors of the previous horizon use the transition probability to produce the substituted case partitions for all continuous state variables x'_{s_i}:

Algorithm 3: `VI`(DC-POMDP, H) $\longrightarrow (V^h, \pi^{*,h})$

```
1  begin
2      V^0 := 0, h := 0
3      while h < H do
4          h := h + 1, Γ^h := ∅
5          foreach a ∈ A do
6              Γ_a^h := ∅
7              O^{h-1} := GenRelObs(Γ^{h-1}, a)
8              foreach o_i ∈ O^{h-1} do
9                  foreach α_j^{h-1} ∈ Γ^{h-1} do
10                     α_j^{h'} = Prime(α_j^h)
11                     // ∀ d_i → d'_i and ∀x_i → x'_i
12                     Γ^h_{a,o_i,j} := o_i ⊗
                       Backup(α_j^{h'}, a)
13                 Γ^h_{a,o} := ⊞ Γ^h_{a,o_i}
14             Γ_a^h := R_a ⊕ γ · Γ^h_{a,o}
15             Γ^h := Γ^h ∪ Γ_a^h
16         V^h := Prune(Γ^h)
17         π^{*,h} := arg max_a Γ_a^h
18         if V^h = V^{h-1} then
19             break // Terminate if early
                   convergence
20
21     return (V^h, π^{*,h})
22 end
```

$$G^2_{close} = \begin{cases} (t'_s > 235) & : -1000 \\ \neg(t'_s > 235) & : 100 \end{cases} \tag{8}$$

$$G^2_{open} = \top : -1$$

4.3 Value iteration for DC-POMDPs

Having defined the two main algorithms to produce the relevant observation partitions and the backup operation, we now explain 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.

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 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: $\Gamma^1 = \{\alpha^1_{1,open}, \alpha^1_{1,close}\}$.

For $h = 2$, for each of the discrete actions, first we generate the relevant set of observations. This has been done for the $close$ action in the previous section. Next for each of the relevant partitions and both α-vectors of the previous Γ set lines 8–15 are executed.

Line 10 will substitute prime values for all state variables. Line 12 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 12 for observations O_2, O_3 and α_1^1, α_2^1 is defined as:

$$\begin{cases} \Gamma^2_{close,O_2,1} = \begin{cases} (t_o > 300) \wedge (t'_s > 235) & : -100 \\ (t_o > 300) \wedge (200 < t'_s < 235) & : 10 \\ (t_o > 300) \wedge (t'_s < 200) & : 0 \end{cases} \\ \Gamma^2_{close,O_2,2} = \begin{cases} (t_o > 300) \wedge (t'_s > 200) & : -0.1 \\ (t_o > 300) \wedge (t'_s < 200) & : 0 \end{cases} \end{cases}$$

$$\begin{cases} \Gamma^2_{close,O_3,1} = \begin{cases} (t_o < 200) \wedge (t'_s > 235) & : -900 \\ (t_o < 200) \wedge (200 < t'_s < 235) & : 90 \\ (t_o < 200) \wedge (t'_s < 200) & : 0 \end{cases} \\ \Gamma^2_{close,O_3,2} = \begin{cases} (t_o < 200) \wedge (t'_s > 200) & : -0.9 \\ (t_o < 200) \wedge (t'_s < 200) & : 0 \end{cases} \end{cases}$$

Next in line 13 we have to take the cross-sum of the two sets which takes every possible combination of observations according to the α-vectors. This means for $O_2 \boxplus O_3$ we have the following vectors:

$$\begin{cases} \begin{cases} (t'_s > 235) & : -1000 \\ (200 < t'_s < 235) & : 100 \\ (t'_s < 200) & : 0 \end{cases} \\ \begin{cases} (t'_s > 235) & : -100.9 \\ (< 200 < t'_s < 235) & : 9.1 \\ (t'_s < 200) & : 0 \end{cases} \\ \begin{cases} (t'_s > 235) & : -900 \\ (200 < t'_s < 235) & : 89.9 \\ (t'_s < 200) & : 0 \end{cases} \\ \begin{cases} (t'_s > 200) & : -1 \\ (t'_s < 200) & : 0 \end{cases} \end{cases}$$

Where the observation partitions $t_o > 300 and t_o < 200$ are inconsistent and therefore omitted from the result. These 4 vectors are added with the two vectors from Γ^2_{close,O_1} to get 2^3 different vectors. All 8 vectors are multiplied by the discount γ and added with the reward in line 14 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.

Algorithm 4: `Prune`$(\Gamma^h) \longrightarrow (V^h)$

```
1  begin
2      V^h := 0 ;
3      foreach α_j^h ∈ Γ^h do
4          α_j^h := LP-Solve(α_j^h) ;
5          foreach α_k^h ∈ Γ^h do
6              if j ≠ k then
7                  V^h := ⊕ Dominance(α_j^h, α_k^h) ;
8
9
10     return (V^h) ;
11 end
```

In line 16 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 V^h. As an example from the set $\Gamma^2_{open,close}$ the two α-vectors of $\alpha_{1,open}(\vec{s}) \geq \alpha_{1,close}(\vec{s})$ are tested:

$$(t_s > 200) \wedge (200 < t_o < 300) \geq (t_s > 150) \wedge (220 < t_o < 280)$$

The second α-vector is omitted from the overall answer since it is dominated by $\alpha_{1,open}$.

5 Empirical Results

6 Related Work

7 Conclusion

References