MILP BASED VALUE BACKUPS IN POMDPS WITH VERY LARGE OR CONTINUOUS ACTION SPACES

Rakshita Agrawal[+]**, Jay H. Lee*** **and Matthew Realff**[+]

+School of Chemical and Biomolecular Engineering, Georgia Institute of Technology
Atlanta, GA 30332, USA
*Department of Chemical and Biomolecular Engineering, Korea Advanced Institute of Science and Technology
Daejoen, Korea

Abstract
Partially observed Markov decision processes (POMDPs) serve as powerful tools to model stochastic systems with partial state information. Since the exact solution methods for POMDPs are limited to problems with very small sizes of state, action and observation spaces, approximate point-based solution methods like Perseus have gained popularity. In this work, a mixed integer linear program (MILP) is developed for calculation of exact value updates (in Perseus and similar algorithms), when the POMDP has very large or continuous action space. Since the solution time of the MILP is very sensitive to the size of the observation space, the concept of post-decision belief space is introduced to generate a more efficient and flexible model. An example is presented to illustrate the concepts and compare the results with those of the existing techniques.

Keywords: Partially Observed Markov Decision Process, Post-Decision Belief State, Bellman Equation, MILP

Introduction

POMDP describes a discrete-time stochastic control process when the states of the environment are partially observed. At any time, the system is in one of the states $s \in S$ where S is a set of all permissible states and is called state space. By taking an action a, the system transitions to the next state $s' \in S$ according to known probability $p(s'|s,a)$ and accrues a reward $r(s,a)$. The next state s' is not completely observed but an observation o may be made, which is probabilistically related to the state s' and action a by $p(o|s',a)$ through stochastic system dynamics. Throughout this paper, symbol $p(.)$ is used to denote probability of a quantity.

More formally, it corresponds to a tuple (S, A, Θ, T, OP, R) where S is a set of states, A is a set of actions, Θ is a set of observations, $T : S\times A\times S \rightarrow [0,1]$ is a set of transition probabilities that describe the dynamic behavior of the modeled environment, $OP : S\times A\times \Theta \rightarrow [0,1]$ is a set of observation probabilities that describe the relationships among observations, states and actions, and R: $S\times A\times S \rightarrow \mathbf{R}^1$ denotes a reward model that determines the reward when action a is taken in state s leading to next state s'. The dependence of reward function on s' is usually suppressed by taking a weighted average over all possible next states ($r(s,a) = \sum_{s'} p(s'|s,a)R(s,a,s')$). Symbols s, s', o and a are used to denote current state, next state, observation and action and belong to sets S, S, Θ and A respectively. $0 \le \gamma < 1$ is the discount rate that discounts the future rewards. The goal is to maximize the discounted sum of rewards over a time horizon, which can be either finite or infinite.

Since the exact solution methods for POMDPs are limited to problems with very small sizes of state, action and observation spaces, approximate solution methods have gained popularity. Notable among these are the point based methods, which consider a fixed or evolving set of prototype belief points instead of considering the entire belief simplex. A particular point based method, PERSEUS (Spaan Matthijs and Vlassis, 2005) favorably makes use of the piecewise linear and convex (PWLC) structure of the value function to speed up convergence. In this work, POMDPs with very large or continuous action space are considered. In the current form of PERSEUS and many other point based methods, presence of continuous actions or very large action space makes it practically impossible to compute the value backups exactly.

In this paper, mathematical programming models are developed to alleviate this difficulty. In section 2, existing literature on solution methods for POMDPs with large/ continuous action space is discussed. In section 3, details of mathematical program to obtain the value backups are discussed along with assumptions on problem structure. The notion of *post-decision* belief state is introduced in section 4. The alternative formulation around post-decision belief state allows for more efficient and flexible computation of the value updates in the presence of large sized observation space. An example is presented in section 5 for the purposes of illustration and comparison.

Related work

Adopting the POMDP notation from (Spaan and Vlassis, 2005), the value backup for a belief point b (for infinite horizon POMDP with discount factor γ) is given by (1) through (4), where $\alpha_n^i, i = 1,2,..|V_n|$ is the set of gradient vectors that characterizes the value function at nth iteration (denoted by V_n). The belief state $b(s)$ represents the probability of being in state s at a given time. $b^{a,o}(s)$ is the belief state at the next time period, which is reached by taking an action a and making an observation o. Infinite horizon POMDP with discounting is used in all illustrations, γ being the discounting factor. Equivalent models can be derived for finite horizon POMDPs with little difficulty.

$$V_{n+1}(b) = \max_{a\in A}\{\sum_{s\in S} r(s,a)b(s) + \gamma \sum_{o\in O} p(o\mid b,a)V_n(b^{a,o})\} \quad (1)$$

$$b^{a,o}(s') = \frac{\sum_{s\in S} p(s'\mid s,a)p(o\mid s',a)b(s)}{\sum_{s'\in S}\sum_{s\in S} p(s'\mid s,a)p(o\mid s',a)b(s)} \quad (2)$$

$$p(o\mid b,a) = \sum_{s'\in S}\sum_{s\in S} p(s'\mid s,a)p(o\mid s',a)b(s) \quad (3)$$

$$V_n(b^{a,o}) = \max_i \sum_{s'\in S} \alpha_n^i(s')b^{a,o}(s') \quad (4)$$

Similar to the fully observable Markov decision processes (FO-MDP or simply MDP), the computation time for each iteration is proportional to $|A|$ when enumeration of all actions is used in the max operation in (1). Additionally, the size of all possible gradient vectors at the nth iteration is $|V_n||A||O|$, where $|V_n|$ is the number of gradient vectors that characterize V_n. Therefore, POMDP with large action and observation spaces prove to be a challenge. It is not surprising then that, to the best of our knowledge, no current solution method claims to compute the max operation exactly for very large or continuous action spaces. We next review a subset of literature that considers POMDPs with very large or continuous action spaces.

Among the available POMDP solution methods, policy search methods are better equipped at handling continuous action spaces. An example is Pegasus (Ng and Jordan, 2000), which estimates the value of a policy by simulating trajectories using a fixed random seed, and adapts its policy in order to maximize the value. Pegasus can handle continuous action spaces at the cost of a sample complexity that is polynomial in the size of the state space. Baxter and Bartlett (2001) propose a policy gradient method that searches in the space of randomized policies, and which can also handle continuous actions. The main disadvantages of policy search methods are the need to choose a particular policy class and the fact that they are prone to local optima.

Thrun (2000) and Spaan and Vlassis, (2005) consider sampling techniques to keep the active size of the action space relatively small for continuous or very large action spaces. In the Monte Carlo POMDP (MC-POMDP) method of Thrun (2000), real-time dynamic programming is applied on a POMDP with a continuous state and action space. In that work, beliefs are represented by sets of samples drawn from the state space, while the values of the Q-functions defined over belief state and action ($Q(b,a)$) are approximated by nearest-neighbor interpolation from a (growing) set of prototype values and are updated by online exploration and the use of sampling-based Bellman backups. In contrast with PERSEUS, the MC-POMDP method does not exploit the piecewise linear and convex structure of the value function. Both methods are problem dependent and may lead to loss of solution quality in certain applications.

Alternatively, by the use of mathematical programming, exact value backups may be ensured in the presence of large or continuous action space. The equivalent mathematical program for (1) along with assumptions is presented in the following section.

Mathematical programming based value updates

Formulation of the mathematical program

The biggest motivation for using a mathematical program to compute the value backup for a belief point b is the fact that the value function for infinite horizon POMDP can be approximated well by a PWLC function (Sondik 1978). The value backup equation is shown in (1). Assuming that the state and observation spaces are finite and with a little abuse of notation, the reward function $r_a(s) = r(s,a)$, state transition probability function $t_a(s,s') = T(a,s,s')$ and observation probability function $op_a(s',o) = OP(s',o)$ are dependent on action a as shown in (5) through (7). Here subscript a suggests dependence on action a; s,s' and o represent the indices of current state, next state and observation respectively.

$$r_a(s) = f_1(a,s) \quad (5)$$
$$t_a(s,s') = f_2(a,s,s') \quad (6)$$
$$op_a(s',o) = f_3(a,s',o) \quad (7)$$

The equivalent mathematical program for (1) can be written as shown in (8) through (13). $\alpha_n(i,s') = \alpha_n^i(s')$ is used for ease of notation.

$$\max_{a\in A}\{\sum_{s\in S} r_a(s)b(s) + \gamma\sum_{o\in O}\sum_{s'\in S}\sum_{s\in S} t_a(s,s')op_a(s',o)b(s)v(o)\} \quad (8)$$

s.t.

$$v(o) = \max_i \sum_{s'\in S} \alpha_n(i,s')b^{a,o}(s') \quad \forall o \quad (9)$$

$$b^{a,o}(s') = \frac{\sum_{s\in S} t_a(s,s')op_a(s',o)b(s)}{\sum_{s'\in S}\sum_{s\in S} t_a(s,s')op_a(s',o)b(s)} \quad \forall s' \quad (10)$$

$$r_a(s) = f_1(a,s) \quad (11)$$
$$t_a(s,s') = f_2(a,s,s') \quad (12)$$
$$op_a(s',o) = f_3(a,s',o) \quad (13)$$

Substituting the value of $b^{a,o}$ from (10) and canceling the term $\sum_{s\in S}\sum_{s'\in S} t_a(s,s')op_a(s',o)b(s)$, the resultant mathematical program is shown in Figure 1. Admittedly, the reward and probability functions in the form of f_1,f_2 and f_3 have not been defined. They are better understood by the illustrative example presented later (see Eq. (30)-(31)).

$$
\begin{array}{lr}
\max_{a\in A}\{\sum_{s\in S} r_a(s)b(s) + \gamma \sum_{o\in O} \tilde{v}(o)\} & \text{M1.1} \\
s.t. & \\
\tilde{v}(o) \geq \sum_{s'\in S}\sum_{s\in S} t_a(s,s') op_a(s',o)\alpha_n(i,s')b(s) \;\; \forall i,o & \text{M1.2} \\
\tilde{v}(o) \leq \sum_{s'\in S}\sum_{s\in S} t_a(s,s') op_a(s',o)\alpha_n(i,s')b(s) + & \\
\qquad M(1-y(i,o)) \;\; \forall i,o & \text{M1.3} \\
\sum_i y(i,o) = 1 \;\; \forall o & \text{M1.4} \\
r_a(s) = f_1(a,s) & \text{M1.5} \\
t_a(s,s') = f_2(a,s,s') & \text{M1.6} \\
op_a(s',o) = f_3(a,s',o) & \text{M1.7}
\end{array}
$$

Figure 1: The mixed integer program (model M1) for determination of the maximizing action for value update

Computational efficiency of the mixed integer formulation

The value backups are computed many times for different belief states in each iteration. The operation is then repeated for multiple iterations. It is therefore imperative that the mathematical program associated with the value backup be computationally efficient and yield near-optimal solutions for each solve. In order to ensure the above two properties, restrictions on the structure of the mathematical program need to be imposed. This limits the applicability of the proposed approach to a certain extent. In general, a linear, quadratic or convex program provides ease of computation. This requires that the stage-wise reward, equations and constraints be a linear, quadratic or convex function of the action. Due to the presence of integer variables, a linear formulation is most suitable.

The the size of the model (number of variables and constraints) is directly proportional to $|S|^2$, $|O|$ and $|V_n|$. Dependence on $|A|$ is implicit. Since f_1, f_2 and f_3 are transition functions, in the absence of logical variables they pose little computational challenge. The size of the model is greatly affected by the number of integer variables, i.e., $y(i,o)$ in this case. This number clearly depends on $|V_n|$ and $|O|$. While $|O|$ comes directly from the model, the size of V_n is governed by a combination of factors. The most important factor is the dimensionality of belief simplex. In terms of PERSEUS, a higher dimensional simplex would require higher number of belief points comprising the prototype belief set and $|V_n| \leq |B|$. Since $|B|$ is highly dependent on the dimension of the state space, methods like value-directed compression (Poupart and Boutilier, 2003) and PCA for belief compression (Roy and Gordon, 2003) may be used to reduce $|S|$ and hence $|B|$. However, in practice $|V_n| << |B|$ (Spaan Matthijs and Vlassis, 2005). Another determinant of $|V_n|$ is the structure of optimal policy. For example, when the decision region is convex, it is possible to approximate the function corresponding to each decision region by only few gradient vectors. In this case, $|V_n|$ would depend on $|A|$.

While approaches to limit the size of $|V_n|$ are dependent on the problem structure, there is a more general and elegant way to resolve the problem of large sized observation spaces. This is addressed in the following section.

Value iteration around post decision belief state

The basic idea

For a general MDP (fully observable), the notion of post decision state applies to problems where the effect of actions and uncertainty on state variable can be separately represented. Since POMDP is equivalent to a continuous state FO-MDP, this concept can be utilized here, given the aforementioned requirement is met. To see this, let the belief state at time t be denoted by b_t. When action a_t is taken, the state can be thought to transition to an intermediate state $\tilde{b}_t^a$ before the next observation is made. Once the uncertain observation o is realized, the system can be in any of $|O|$ next belief states where $|O|$ is the size of uncertainty, i.e., the number of possible next beliefs. This is schematically shown in Figure 2. Circles represent the more popular pre-decision belief state b_t, b_{t+1} etc. and squares represent the intermediate state $\tilde{b}_t^a$ that captures the effect of action only. This is referred to as *post decision belief state*.

In the context of POMDPs, solution using the post-decision state approach is possible when the observation probabilities do not depend on action a. The actions that affect observation probabilities may be made part of the state. The transition from regular belief state b_t to $\tilde{b}_t^a$ then is simply given by $t_a b_t$.

It is to be noted that although the effect of action on underlying states $s\in S$ may be prone to uncertainty, the belief state transition ($t_a b_t$) is *always* deterministic.

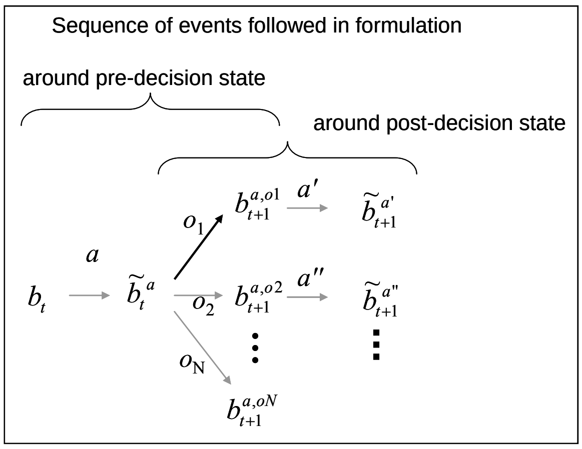

Figure 2: A schematic of pre-decision state to post-decision state and again to pre-decision state

Having obtained the post decision belief state, |*O*| (pre-decision) belief states may be obtained at the next time step. The transition is governed by the observation probabilities $p(o|b_t)=p(o|\tilde{b}_t^a)$ for all $o\in O$. This two step transition is shown in Figure 2. Intuitively, starting with a post decision state $\tilde{b}^a$, |*O*| possible next states $b^{a,o}\ \forall o$ are obtained. For each of the states $b^{a,o}\ \forall o$, a maximizing action *a'* would determine the next post-decision belief state $\tilde{b}^{a',o}$. Consequently, the value iteration update takes the form shown in (15) through (19). V^a represents value function around post-decision belief state $\tilde{b}^a$. For notational ease, the dependence of action *a'* and next post-decision belief state $\tilde{b}^{a'}$ on *o* is suppressed in future illustrations. The details on the derivation of value iteration equation around post-decision state can be found in (Powell 2007) for a general MDP.

In (15), it is to be noted that the expectation over observations is outside the max operator. This removes the dependence of the size of the MILP on |*O*|. In addition, we can show that the structure of V^a can be shown to be still PWLC (Powell 2007)

In the following sections, we derive the value and gradient update equations around post-decision belief state and then present pros and cons of using this method over regular value iteration around pre-decision state.

Derivation of backup equations and formulation of math program

The equations for value and gradient backups around post decision belief states are derived on similar lines as shown in (Spaan and Vlassis, 2005). The value iteration step for the post decision state variable is given by (15) through (19).

$$V_{n+1}^a(\tilde{b}^a)=\sum_o p(o\,|\,\tilde{b}^a,a)\max_{a'}\{\sum_s r(s,a')b^{ao}(s) + \gamma V_n^a(\tilde{b}^{a'}(s'))\} \quad (15)$$

where

$$b^{ao}(s)=\frac{\tilde{b}^a(s)p(o\,|\,s,a)}{\sum_s \tilde{b}^a(s)p(o\,|\,s,a)} \quad (16)$$

$$\tilde{b}^{a'}(s')=\sum_s b^{ao}(s)p(s'\,|\,s,a') \quad (17)$$

$$V_n^a(\tilde{b}^{a'}(s'))=\max_i \sum_{s'} \tilde{b}^{a'}(s')\alpha_n^i(s') \quad (18)$$

$$p(o\,|\,\tilde{b}^a,a)=\sum_s p(o\,|\,s,a)\tilde{b}^a(s) \quad (19)$$

Substituting (16) through (19) in (15)

$$V_n^a(\tilde{b}^a)=\sum_o p(o\,|\,\tilde{b}^a,a)\max_{a'}\{\frac{\sum_s r(s,a')p(o\,|\,s,a)\tilde{b}^a(s)}{\sum_s p(o\,|\,s,a)\tilde{b}^a(s)}+ \gamma\max_i\sum_{s'}\sum_s\frac{p(o\,|\,s,a)p(s'\,|\,s,a')\tilde{b}^a(s)\alpha_n^i(s')}{\sum_s p(o\,|\,s,a)\tilde{b}^a(s)}\} \quad (20)$$

Since $p(o\,|\,\tilde{b}^a,a)$ is independent of a', it is taken out of max and cancelled with the numerator. This implies that

$$V_{n+1}^a(\tilde{b}^a)=\sum_o \max_{a'}\{\sum_s r(s,a')p(o\,|\,s,a)\tilde{b}^a(s)+ \gamma\max_i\sum_{s'}\sum_s p(o\,|\,s,a)p(s'\,|\,s,a')\tilde{b}^a(s)\alpha_n^i(s')\} \quad (21)$$

Using

$$g_{a'o}^i(s)=\sum_{s'} p(o\,|\,s,a)p(s'\,|\,s,a')\alpha_n^i(s') \quad (22)$$

$$and\ \ T^{a'o}(s)=r(s,a')p(o\,|\,s,a) \quad (23)$$

and the identity

$$\max_{\{y_j\}_j} x.y_j = x.\arg\max_{\{y_j\}_j} x.y_j \quad (24)$$

$$V_{n+1}^a(\tilde{b}^a)=\sum_o \max_{a'}\{\sum_s T^{a'o}(s)\tilde{b}^a(s)+ \gamma\tilde{b}^a.\arg\max_{\{g_{a'o}^i\}_i} <\tilde{b}^a,g_{a'o}^i>\} \quad (25)$$

$$G^{a'o}(s)=T^{a'o}(s)+\gamma\arg\max_{\{g_{a'o}^i\}_i} <\tilde{b}^a,g_{a'o}^i> \quad (26)$$

$$V_{n+1}^a(\tilde{b}^a)=\sum_o \{\tilde{b}^a.\arg\max_{\{G^{a'o}\}_{a'}} <\tilde{b}^a,G^{a'o}>\} \quad (27)$$

and

$$\alpha_{n+1}^{\{\tilde{b}^a\}}=\sum_o \arg\max_{\{G^{a'o}\}_{a'}} <\tilde{b}^a,G^{a'o}> \quad (28)$$

(27) and (28) give the value backup and gradient vector backup for the post decision belief state variable respectively.

The mathematical program for (25) for a given observation and belief state $\tilde{b}^a$ is shown in Figure 3. Evidently, the binary variables *y*(*i*) for *i*=1,2,. |V_n| and variable *v* do not depend on observation *o*. However the model has to be solved multiple times to obtain the backup. The number of times the model is to be solved is given by *o_size*≤|*O*|, where *o_size* is the number of observations for which $p(o|\tilde{b}^a)>0$.

Policy determination Similar to the pre-decision state case of section 3.2, there are two possibilities to determine the optimal action for a belief state *b*, i.e., (i) Storing the maximizing action(s) associated with each gradient vector and (ii) Solving one step look-ahead problem on-line. However, there are multiple actions associated with a gradient backup ($a'^{\{o\}}\ \forall o$). Therefore the action associated with each observation needs to be cached. This results in higher memory requirement for storing the ε-optimal policy for the post-decision state, as compared to that for the pre-decision state. For the look-ahead design on the other hand, the MILP needs to be solved for the belief state pertaining to the current observation only. This is computationally less expensive than the look-ahead design for solution around pre-decision belief state.

$$\boxed{\begin{aligned}
&\max_{a\in A}\{\sum_{s\in S} r_{a'}(s)b^{a,o}(s)+\gamma v(o)\}\\
&s.t.\\
&v(o)\geq \sum_{s'\in S}\sum_{s\in S} t_a(s,s')\alpha_n(i,s')b(s) \quad \forall i\\
&v(o)\leq \sum_{s'\in S}\sum_{s\in S} t_a(s,s')\alpha_n(i,s')b(s)+M(1-y(i)) \quad \forall i\\
&\sum_i y(i)=1\\
&r_a(s)=f_1(a,s) \quad \forall s\\
&t_a(s,s')=f_2(a,s,s') \quad \forall s
\end{aligned}}$$

Figure 3: The MILP for determination of the maximizing action for value update for a post-decision belief state

Comparison with value updates around pre-decision belief states

While using MILP based value updates the two methods can be compared along following avenues:

i) Complexity of MILP problem – In the post decision state formulation (referred to as 'post-formulation' hereafter), several smaller MILPs are solved for one value backup as opposed to solving one large MILP for pre-decision state formulation ('pre-formulation'). The former almost always works better if the input/output operations between the MILP solver (e.g., CPLEX) and regular solution platform (e.g., MATLAB) are not as time consuming as the optimization itself.

ii) Policy determination in real time – The formulation around pre-decision state allows for storing optimal action with each gradient vector that characterizes the ε-optimal value function. For formulation around post-decision belief state, optimal action needs to be stored for each gradient vector and each observation. This increases the memory requirement for the latter. This may not be feasible when the observation space is very large. However, the look-ahead design for policy determination is faster for the post-formulation due to lower complexity of the associated MILP.

iii) Handling very large or continuous observation spaces – While pre-formulation is limited to small observation spaces, the MILP technique for value updates based on post-formulation is on par with enumeration based methods. When observation space is very large or continuous, Hoey and Poupart (2005) consider creating sets of observations for which b leads to the same future belief state $b^{a,o}$. This effectively makes the observation space discrete. Such manipulation of observation probabilities is better achieved outside the confines of a mathematical program, thus making the post-formulation more attractive.

Aside from (ii) above, it is easy to see that the two formulations are similar in terms of computational complexity when enumeration of action space is used for value backups. This consideration excludes the fact that post formulation allows for parallel processing of max operation.

Illustrative example

The example in this section contains continuous actions and the POMDP is formulated around pre-decision belief state. For simplicity, the observation probabilities are assumed independent of actions.

In order to illustrate the concept of using mathematical programming for value backups, a simple problem with two states is considered first. A hypothetical machine can be in one of two states (s_1 and s_2) at any time. The system probabilistically transitions between the two states.

Rewards R_1 and R_2 are received when system is in state s_1 and s_2 respectively and R_1>R_2. This implies that s_1 is more desirable state than state s_2. There are two possible actions a_1 and a_2 which affect the probabilities of state transition as shown below. $a_1\in(0,1)$ and $a_2\in(0,1)$ are continuous and bounded. While a higher value of a_1 helps the system remain in state 1, a higher value of a_2 ensures it's returning to state s_1 from state s_2. These actions can be thought of as routine preventive and corrective actions to make sure the equipment is in state s_1, e.g. cleaning, lubrication etc. The tasks are scaled to obtain the bounds of 0 and 1.

$$S=\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \quad A=\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad O=\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \qquad (29)$$

$$T_a=\begin{bmatrix} 0.5(1+a_1) & 0.5(1-a_1) \\ a_2 & 1-a_2 \end{bmatrix} \qquad (30)$$

The (state dependent) unit costs of taking action a_1 and a_2 in state s_1 are C_{11} and C_{12} respectively. Similarly, the unit costs of taking action a_1 and a_2 in state s_2 are C_{21} and C_{22} respectively with C_{12}>C_{11} and C_{22}>C_{21} .This ensures that the cost of keeping the system in state s_1 is lower than bringing it back to s_1, when it has transitioned to s_2. Accurate state observation is made with probability

0<β<1. The resulting observation and reward matrices are as shown below:

$$OP=\begin{bmatrix}\beta & 1-\beta \\ 1-\beta & \beta\end{bmatrix} \quad R_a=\begin{bmatrix}R_1-C_{11}a_1-C_{12}a_2 \\ R_2-C_{21}a_1-C_{22}a_2\end{bmatrix} \tag{31}$$

Finally, there are system and budget constraints of the form shown in (32) and (33), respectively. While the former specify system requirements, e.g., a certain mix of cleaning fluids from the two actions, the latter represent limits on total expenditure. Since cost of actions is state dependent, the probabilities of being in state s_1 and s_2, i.e., $b(s_1)=b_1$ and $b(s_2)=b_2$, are also a part of the constraints.

$$A_1a_1 + A_2a_2 + A_3 \le 0 \tag{32}$$
$$B_{11}b_1a_1 + B_{12}b_1a_2 + B_{21}b_2a_1 + B_{22}b_2a_2 + B_3 \le 0 \tag{33}$$

The parameter values considered for this study are shown in Table 1.

Table 1: Parameter values for the system with two-continuous actions

Parameter	β	R_1	R_2	C_{11}	C_{12}	C_{21}	C_{22}	A_1
Value	0.9	2	0	0.1	0.15	0.3	0.05	0.5088
Parameter		A_3	B_{11}	B_{12}	B_{21}	B_{22}	B_3	A_2
Value		-1	3.1185	9.355	3.0736	1.025	-1	1.325

Optimal policy and value function

The performance of the converged value function and policy for both approaches are shown in Table 2. While the pre-decision formulation is used for the MILP based approach, a uniform grid of 0.1 is used for the enumeration based approach to generate a finite action space. The enumeration based method convergences in an order of magnitude less time than the MILP based method. However, the performance of the enumeration based method is worse than that of the MILP solution. This is attributed to the discretization of the action space.

Scalability To study how the solution time scales with the problem size and understand whether the value gap depends on the problem size, two additional experiments are performed: (i) a system with three possible states and three actions, (ii) a system with four states and four actions. Similar to the two-action system above, the states are discrete and all actions are continuous and range from 0 to 1. The probability transition matrix is a linear function of actions and the constraints follow the same linear structure as (29) and (30). The corresponding results are reported in Table 2. It is observed that the performance gap widens as the problem size grows.

We have also applied the post-formulation based MILP value backup method to a realistic network flow problem, which showed significant improvements in solution time and / or quality compared to the enumeration approach as the number of nodes in the network becomes larger. This result is not presented here due to the space limitation, however.

References

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research, 15:319-350, 2001.

S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. Artificial Intelligence, 99(1):21-71, 1998.

T. J. Spaan Matthijs and N. Vlassis, Perseus: Randomized point-based value iteration for POMDPs, Journal of Artificial Intelligence Research, 24: 26, 2005.

J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In Proc. Int. Joint Conf. on Artificial Intelligence, 2005.

W. B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley-Interscience, 2007.

E. J. Sondik, The optimal control of partially observable Markov processes over the infinite horizon: discounted costs, Operations Research, 26(2): 282-304, 1978.

A. Y. Ng and M. Jordan, PEGASUS: A policy search method for large MDPs and POMDPs. In Proc. of Uncertainty in Artificial Intelligence, 2000.

P Poupart, C Boutilier, Value-directed compression of POMDPs, Advances in Neural Information Processing Systems, 2003.

N Roy, G Gordon, Exponential family PCA for belief compression in POMDPs, Advances in Neural Information Processing Systems, 2003.

Table 2. Results for the problems with continuous actions

Problem	$\|A\|$	**Performance**		$\|V_n\|$		**Convergence time**	
		MILP	Enum	MILP	Enum	MILP	Enum
2 actions	121	13.28 ±1.	13.08 ±1.5	6	7	16.0	0.62
3 actions	133	16.71 ±2.	14.67 ±3.2	15	11	46.7	29.45
4 actions	146	15.02 ±1	10.27 ±3.6	16	159	108.	1180.25