Computers and Chemical Engineering 56 (2013) 101–113

Contents lists available at SciVerse ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

MILP based value backups in partially observed Markov decision processes (POMDPs) with very large or continuous action and observation spaces

Rakshita Agrawal[a], Matthew J. Realff[a], Jay H. Lee[b,*]

[a] School of Chemical and Biomolecular Engineering, Georgia Institute of Technology, Atlanta, GA 30022, USA
[b] Department of Chemical and Biomolecular Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea

ARTICLE INFO

Article history:
Received 22 February 2013
Received in revised form 24 April 2013
Accepted 25 April 2013
Available online 27 May 2013

Keywords:
Markov decision processes
Dynamic programming
Mathematical programming
Partial observation
Network reliability

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 involving a flow network is presented to illustrate the concepts and compare the results with those of the existing techniques.

1. 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 discrete 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 probabilistically 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. The sets of all permissible actions and observations are called action and observation space, respectively. The objective is to find a control policy mapping the probability distribution across the states (called *belief state*) to action such that the expected value of the total rewards accrued over a time horizon is maximized.

Many real-world decision/control problems are characterized by probabilistic state transitions, partially observed states, and a reward/cost functions to be maximized/minimized over a finite or infinite time horizon. Therefore POMDPs have appeared in various applications including machine maintenance, structural inspection, elevator control, robotics, dialog control, and even population control in fishery industries (Cassandra, 1998). The main obstacle to wider use, however, has been the lack of computationally efficient algorithms for the value update.

Exact solution for POMDP requires solving the Bellman's optimality equation (Bellman, 1957) formulated with respect to the belief state. 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. A saving grace for POMDP is that the optimal value function is piecewise linear and convex. Hence, the point based methods, which consider a fixed or evolving set of prototype belief points instead of considering the entire belief simplex, have been popular. A particular point based method called PERSEUS (Matthijs & 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.

There is some 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 & 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

* Corresponding author. Tel.: +82 42 350 3926; fax: +82 42 350 3910.
E-mail addresses: jayhlee@kaist.ac.kr, eva@kt.dtu.dk (J.H. Lee).

0098-1354/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compchemeng.2013.04.020

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; this method 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 (1998) and Matthijs 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 (1998), 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 PWLC 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 and/or observation space. This paper seeks to make two contributions: (i) First, mathematical programming models are developed to compute exact value backups associated with PERSEUS in presence of very large of continuous action spaces. (ii) Alternative formulation around post decision belief state is developed to allow for more efficient and flexible computation of the value updates in the presence of large sized observation space. The two algorithms are illustrated by example problems and results are compared with those from traditional techniques.

The paper is organized as follows: In Section 2, we describe POMDPs and the point based solution method PERSEUS which is the underlying algorithm we are using to solve POMDPs. We also motivate the problem by using an illustrative flow network problem and describe its formulation as POMDP. In Section 3, we develop the mathematical program to compute the value backups for PERSEUS. In Section 4, we introduce the notion of post decision belief state and reformulate value backups around it. In Section 5, we discuss application of developed mathematical programs to two illustrative problems and show results in terms of convergence times and solution quality.

2. POMDP description and motivating example

2.1. POMDP description

POMDP 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: $S \times A \times S \rightarrow R^1$ 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. Since the state s at each time is not observed directly but indirectly through o, one does not know the current state with certainty. Hence, the information about the state is represented by belief state $b(s)$, which represents the probability of being in state s at a given time. $b(s)$ for all $s \in S$ must be defined at each time and the vector containing the entire probability distribution over the state space is called the belief state vector to be denoted by b hereafter. $0 \leq \gamma < 1$ is the discount rate that discounts the future rewards. The goal is to find a control policy mapping the belief state vector to a that maximizes the discounted sum of rewards over a time horizon, which can be either finite or infinite (in our case) as shown in (1).

$$\Pi^* = arg\left(\max_{a_t}\sum_{t=1}^{\infty}\gamma^{t-1}r(s_t,a_t)\right) \tag{1}$$

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.

The optimal policy for the above is defined by the following so called Bellman's optimality equation written with respect to the belief state vector.

$$V(b) = \max_{a\in A}\left\{\sum_{s\in S}r(s,a)b(s) + \gamma\sum_{o\in O}p(o|b,a)V(b^{a,o})\right\} \tag{2}$$

$$b^{a,o}(s') = \frac{\sum_{s\in S}p(s'|s,a)p(o|s',a)b(s)}{\sum_{s'\in S}\sum_{s\in S}p(s'|s,a)p(o|s',a)b(s)} \tag{3}$$

$$p(o|b,a) = \sum_{s'\in S}\sum_{s\in S}p(s'|s,a)p(o|s',a)b(s) \tag{4}$$

whereas 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.

Typically, the above is solved through iteration in the following manner:

$$V_{n+1}(b) = \max_{a\in A}\left\{\sum_{s\in S}r(s,a)b(s) + \gamma\sum_{o\in O}p(o|b,a)V_n(b^{a,o})\right\} \tag{5}$$

The above equation is very difficult to handle from a numerical standpoint as $V(b)$ is an arbitrary function of the continuous belief state vector b. Note that the right side involves maximization of an expectation quantity.

2.2. Point based solution method for POMDP

A saving grace for POMDP is that the optimal value function is known to be piecewise linear and convex. Based on this property, point based methods like PERSEUS have become popular.

Adopting the POMDP notation from (Matthijs & Vlassis, 2005), in the point based methods, the value function that appears in the value backup of Eq. (5) takes the form of

$$V_n(b^{a,o}) = \max_{i}\sum_{s'\in S}\alpha_n^i(s')b^{a,o}(s') \tag{6}$$

In the above, $\alpha_n^i, \quad i = 1, 2, \ldots |V_n|$ is the set of gradient vectors that characterizes the value function at nth iteration (denoted by V_n). The dimensionality of the gradient vectors is $|S|$, the size of the underlying state space. $\alpha^i{}_n$ (s') represents the scalar element of the gradient vector $\alpha^i{}_n$ corresponding to the state s'. Similar to the fully observable Markov decision processes (FO-MDP or simply MDP), for every iteration, the computation time is proportional to $|A|$ when enumeration of all actions is used. This is attributed to the max operation in (5). Additionally, the size of all possible gradient vectors at the $n+1$th 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

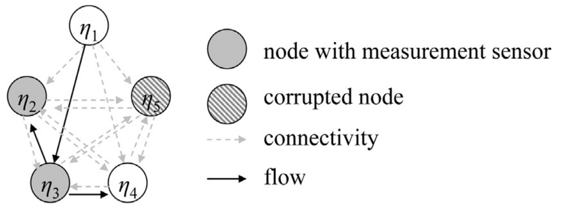

Fig. 1. A five-node network flow problem.

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 and observation spaces.

2.3. *Motivating example: flow network*

2.3.1. *Description*

In order to motivate the discussion on large size POMDPs, let us consider a flow network problem shown in Fig. 1. In this section we describe the problem and formulate it as a POMDP. In Section 5.2 we solve this problem using the algorithm developed as part of this work.

The general structure of a network consists of nodes and edges. The nodes, shown as circles in Fig. 1, facilitate the accumulation, production or consumption of materials or information while the edges facilitate flow of these quantities from one node to the other. The edges are shown as arrows originating from the source node and pointing to the destination or recipient node. Many studies (Berry, 2005; Rico-Ramirez, Frausto-Hernandez, Diwekar, & Hernandez-Castro, 2007) have been conducted for network design, i.e., determining the connectivity of the nodes. In other words, network design determines the existence of edges through which it is possible to transport material, information or both. The nodes of certain networks are amenable to contamination or corruption, e.g., computer networks can get infected with virus/bugs, food and water networks are prone to chemical or biological contamination and electrical networks are amenable to outages. Aside from origination, the contamination or infection spreads as the material or data flows from one node to the other. Therefore, it is imperative to track down the infected node and redirect the network flows. To this end, sensor network design studies have been performed to determine optimal locations to install measurement sensors. However, in the face of budget constraints, all the nodes and edges cannot be inspected.

Whereas network design and sensor allocation are one time decisions requiring substantial investments, network flow decisions are dynamic decisions that must be taken at each time so as to minimize the expected spread of the contaminant. The network flow decisions differ from network design decisions in that the latter determine which two nodes *can* have flow/connectivity between them by the presence of an edge, while the flow decisions determine whether or not to use an existing edge for flow. In Fig. 1, the light dashed lines show network connectivity and solid lines show a flow configuration. Especially in computer, electrical and water networks, the alteration in network flows is speedy and comes at very low cost.

2.3.2. *Model details*

Fig. 1 shows a network flow problem with five nodes (η_1, η_2, η_3, η_4, η_5). The network is fully connected in that all nodes are connected to each other except node η_1. η_1 serves as the source node, which has no incoming streams. The shaded circles represent ones with measurement sensors and hashed circles indicate corrupted nodes. Hence, measurement sensors are installed at nodes η_2 and η_3. This can be interpreted as saying that all outgoing streams emanating from nodes η_2 and η_3 are tested. The measurement is prone to type I errors (Lee and Unnikrishnan, 1998), such that the presence of a contaminant is detected with probability 0.9. If a node is connected to any upstream node, a reward of C_P units weighted by the population density at that node is received. However, if the node is infected, no reward is received. There are two ways in which a node can get infected: (i) origination of infection at that node and (ii) propagation of contaminant from an upstream node. Contamination is originated at a node with predetermined probabilities p_i for $i = 2, 3, 4, 5$. At each time, at most one node may be infected by origination of contaminant at that node. Alternatively, if an upstream node is infected, the recipient node would get infected at the next time period. It is assumed that the source node η_1 is never contaminated.

A node once contaminated, remains so unless a clean-up is performed. The clean-up has an associated cost C_r, which is assumed to be independent of the node being cleaned. Finally, there is a cost $C_{flow}(i,j)$ associated with flow of material/information in the network. The cost depends on the source node i and recipient node j. The objective is to determine the optimal flow configuration and clean-up strategy at each time, so as to maximize average (infinite horizon discounted) profit. The knowledge of the nodes being contaminated is not complete.

We begin with a POMDP formulation of the network flow problem described above, and analyze the problem size. This is followed by the MILP formulation of the value backup operation.

2.3.3. *Formulation as POMDP*

While the state and action spaces are easily represented, the transition function is complicated for this problem. The POMDP model is shown below:

2.3.3.1. *State.*

$$s(i,t) \in \{0,1\} \quad i = 2, 3, \ldots, N$$
$$|S| = 2^{(N-1)} \tag{7}$$

The size of state space is given by (7) where N is the number of nodes in the network. $s(i,t) = 1$ means the ith node is contaminated at time t and $s(i,t) = 0$ otherwise.

2.3.3.2. *Action.*

$$a_{flow}(i,j,t) \in \{0,1\} \quad i = 1, 2, \ldots, N; \quad j = 2, 3, \ldots, N$$
$$a_{repair}(i,t) \in \{0,1\} \quad i = 2, 3, \ldots, N$$

The term $a_{flow}(i,i)$ for $i = 2,3,\ldots,N$ is the node activation term. $a_{flow}(i,i) = 1$ implies that node i is connected indicating that it has a positive inflow from some other node in the network. $a_{flow}(1,1,t)$ is always 1.

$$|A| = 2^{N(N-1)} + 2^{(N-1)} \tag{8}$$

The expression for the size of the action space in (8) suggests a very large action space for even small values of N. However, many actions are not feasible. E.g.,

(i) A node with no inflow cannot have any outflow or accumulation.
(ii) A node with positive inflow must have either accumulation or outflow.
(iii) A node can be either upstream or downstream of another node but not both.
(iv) Each node can have at most one supply node.

The above constraints reduce the size of the action space considerably.

2.3.3.3. State transition function. Given the state and action at the current time, the state at the next time may be determined by the following relations:

(i) Contamination by origination – A node i is contaminated by origination depending on the contamination probability p_i. This is shown in (9), where $p(t+1)$ is the realization of probability of origination of contamination and $\delta(\cdot)=1$ if the condition within the parenthesis is met.

$$s(i,t+1)=\delta(p(t+1)\le p_i)\quad \text{for} i=2,3,\dots,5 \tag{9}$$

(ii) Contamination by propagation – If a source node k is contaminated, recipient node i will contaminated at the next time period (10).

$$s(i,t+1)=s(k,t)a_{flow}(k,i,t)$$
$$\text{for} i=2,3,\dots,5;\ k=2,3,\dots,5 \tag{10}$$

(iii) Carried over contamination – If node i is contaminated and not repaired at the current time, it will be contaminated at the next time period (11).

$$s(i,t+1)=\max\{s(i,t)-a_{repair}(i,t),0\}\quad \text{for} i=2,3,\dots,5 \tag{11}$$

As the transition matrix for contamination by origination (for point (i) above) is only governed by contamination probability at each node, it is a static matrix independent of current state and action. Let us designate this constant matrix as map_1. The transition matrices that account for points (ii) and (iii) above need to be determined for each state action pair. Let the matrix that combines the effects of factors (ii) and (iii) be designated by map_2. The post decision belief state is given by (12).

$$b^a(s')=\sum_{s\in S} b(s)map_1(s,s')map_2{}^{\{a\}}(s,s') \tag{12}$$

We will revisit matrices map_1 and map_2 in more detail in Section 5.2.1.

2.3.3.4. Observation space and observation probability matrix.

$$\begin{aligned} &o(i,t)\in\{0,1\}\quad i\in M\\ &|O|=2^{|M|}\end{aligned} \tag{13}$$

The size of observation space is given by (13) where M is the set of nodes with measurement sensors. It is assumed that all measurement sensors can sense the presence of a contaminant 90% of the time. The observation probability matrix reflects this fact. For example, if the measurement sensors are installed at nodes η_2 and η_3, the possible observations are:

$$\{o(2,t),o(3,t)\}=\{0,0\};\quad \{1,0\};\quad \{0,1\};\quad \{1,1\}$$

When nodes η_2 and η_3 are both contaminated, i.e., $s(2,t)=s(3,t)=1$, the observation probabilities are given by:

$$p(\{o(2,t),o(3,t)\}=\{0,0\})=(1-0.9)^2$$
$$p(\{o(2,t),o(3,t)\}=\{1,0\})=0.9(1-0.9)$$
$$p(\{o(2,t),o(3,t)\}=\{0,1\})=(1-0.9)0.9$$
$$p(\{o(2,t),o(3,t)\}=\{1,1\})=0.9^2$$

The observation probabilities for the other states may be calculated in a similar manner.

2.3.3.5. Profit/reward function. The profit function at time t is composed of the following terms

Reward for active, non-contaminated nodes – This is given by the first term in (14)

Cost of flow and repair actions – This is given by the second and third terms in (14).

$$\begin{aligned} r_t(s,a)=\ &C_P\sum_i w_i a_{flow}(i,i,t)s(i,t)\\ &-\sum_j\sum_i C_{flow}(i,j)a_{flow}(i,j,t)\\ &-\sum_i C_{repair}(i)a_{repair}(i,t)\end{aligned} \tag{14}$$

Recall that $a_{flow}(i,i)=1$ means that ith node is active. The value backups using enumeration are straightforward once the transition matrices are developed for each action.

3. Mathematical programming based value updates

3.1. 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 (2). Assuming that the state and observation spaces are finite and with a little abuse of notation, let us use the following for reward function $r_a(s)=r(s,a)$, state transition probability function $t_a(s,s')=T(a,s,s')$ and observation probability function $op_a(s',o)=OP(s',a,o)$. Here subscript a suggests dependence on action a; s, s' and o represent the indices of current state, next state, and observation, respectively.

The equivalent mathematical program for (2) can be written as shown in (15) through (17). $\alpha_n(i,s')=\alpha_n^i(s')$ is used for ease of notation.

$$\max_{a\in A}\left\{\sum_{s\in S} r_a(s)b(s)+\gamma\sum_{o\in O}\sum_{s'\in S}\sum_{s\in S} t_a(s,s')op_a(s',o)b(s)v(o)\right\} \tag{15}$$

s.t.

$$v(o)=\max_i\sum_{s'\in S}\alpha_n(i,s')b^{a,o}(s')\quad \forall o \tag{16}$$

$$b^{a,o}(s')=\frac{\sum_{s\in S}t_a(s,s')op_a(s',o)b(s)}{\sum_{s'\in S}\sum_{s\in S}t_a(s,s')op_a(s',o)b(s)}\quad \forall s' \tag{17}$$

Fig. 2(a) represents (16) for a query point b^Q for a two state problem. The system can be in one of two states (s_1 and s_2) at any time. The x-axis represents the probability of being in state s_2, y-axis shows the value function estimate during nth iteration. This is represented by means of gradient vectors α_n^i (which is a vector of dimensionality $|S|=2$ in this case). For a query point $b^{a,o}=b^Q$, (15) takes the maximum of points $p_{\alpha i}$ $i=1,2,\dots|V_n|$. In order to accomplish this, a set of constraints (18) is used to define the feasible region using a proxy variable $\tilde{v}(o)$. The feasible region is shown by the shaded area in Fig. 2(b).

$$\tilde{v}(o)\ge\sum_{s'\in S}\alpha_n(i,s')b^{a,o}(s')\quad \forall i,o \tag{18}$$

However, if $\sum_o\tilde{v}(o)$ is maximized in the objective function, it would make the problem unbounded. In order to set $v(o)$ equal

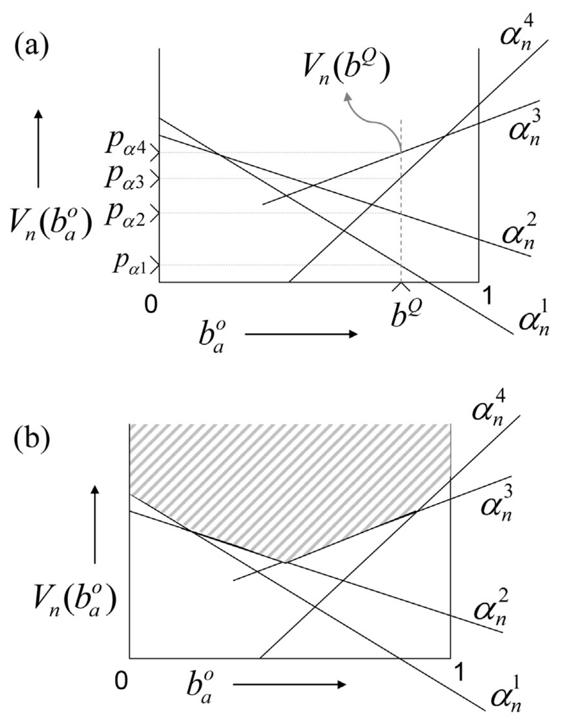

Fig. 2. (a) Value function calculation at a query point, (b) feasible region for the MILP model for value update.

to $\max_i \sum_{s' \in S} \alpha_n(i,s') b^{a,o}(s')$, additional sets of constraints of (19) and (20) are introduced by using a set of binary variables $y(i,o)$.

$$\tilde{v}(o) \le \sum_{s' \in S} \alpha_n(i,s') b^{a,o}(s') + M(1 - y(i,o)) \quad \forall i, o \tag{19}$$

$$\sum_i y(i,o) = 1 \qquad \forall o, y(i,o) \in \{0,1\} \tag{20}$$

For those gradients not corresponding to the maximizing piece for a given $b^{a,o}$, $y(i,o)$ will be 0. Here M must be chosen large enough that constraint (19) is not active for all such i. In a general case, this is achieved by setting $M = \max_i\{\max_{s'} \alpha_n(i,s')\}$. Therefore, constraint sets (18) through (20) exactly model Eq. (16), when the requirement of (21) is met.

$$M \ge \max_i \sum_{s' \in S} \alpha_n(i,s') b^{a,o}(s') - \sum_{s' \in S} \alpha_n(i,s') b^{a,o}(s') \quad \forall i, o \tag{21}$$

Substituting the expression for $b^{a,o}(s')$ from (17) into (18) and (19) and canceling the term $\sum_{s \in S}\sum_{s' \in S} t_a(s,s') op_a(s',o) b(s)$, the mathematical program of Fig. 3 results.

$$\max_{a \in A}\{\sum_{s \in S} r_a(s) b(s) + \gamma \sum_{o \in O} \tilde{v}(o)\} \qquad \text{M1.1}$$

$s.t.$

$$\tilde{v}(o) \ge \sum_{s' \in S}\sum_{s \in S} t_a(s,s') op_a(s',o) \alpha_n(i,s') b(s) \quad \forall i, o \qquad \text{M1.2}$$

$$\tilde{v}(o) \le \sum_{s' \in S}\sum_{s \in S} t_a(s,s') op_a(s',o) \alpha_n(i,s') b(s) + M(1 - y(i,o)) \quad \forall i, o \qquad \text{M1.3}$$

$$\sum_i y(i,o) = 1 \quad \forall o \qquad \text{M1.4}$$

Fig. 3. The mixed integer program (model M1) for determination of the maximizing action for value update.

3.2. Computational efficiency of the mixed integer formulation

The value backups are computed for several 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, bilinear, quadratic or convex program has the necessary structure to permit the computation of near optimal solutions in reasonable time. 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. We present requirements on the model for the mathematical program to be linear. Similar analysis may be carried out for quadratic or convex programs.

We first list the requirements on reward and probability functions for the mixed integer program to be linear and then provide ways to work around some of the requirements. It is easy to see that linearity of the mixed integer model is determined by the structure of functions in $r_a(s)$, $t_a(s,s')$, and $op_a(s',o)$. Let these functions be denoted by $f_1(a)$, $f_2(a)$, and $f_3(a)$, respectively. Then, the following restrictions are placed:

(i) f_1 is a linear function of action a
(ii) $f_2 f_3$ is a linear function of a

Condition (ii) essentially implies that at least one of the functions (f_2 and f_3) may not depend on a. This is imposed by constraints M1.2 and M1.3. Additionally, when functions f_1, f_2 or f_3 involve use of binary variables to represent logical constraints, the linearity of the model may be affected. Finally, when the system state has multiple dimensions, it may be necessary to include the entire state description as opposed to the state indices s and s' in order to determine stage-wise reward and probabilities. This aspect is covered in illustration 2.

Whereas condition (i) is true of many real world systems, at the first glance, condition (ii) appears to be overly restrictive to warrant successful application of the method. However, a closer look reveals that many real world systems may be modeled while satisfying condition (ii). It is generally seen that state transition probabilities depend strongly on action choice, i.e., f_2 is a function of a. Observation probabilities (if at all) depend on discrete actions like whether a sensor measurement is taken. Alternatively, the discrete decisions that affect observation probabilities may include which or how many sensors are used for measurement/taking the observation. These discrete decisions can be made a part of state description leading to observation probabilities depending only on state s' and observation o, but independent of action a.

The specific forms of the reward and probability functions f_1, f_2 and f_3 are not given here. These functions are better understood by illustrative examples presented in Section 5.

3.3. Implementation and policy determination

3.3.1. POMDP solution algorithm

The mathematical program shown in Fig. 3 is a general model to determine value backup for a belief point b when the parameterized form of value function is used. PERSEUS (Matthijs & Vlassis, 2005) is a point based algorithm which directly uses the parametric form of the value function by maintaining a finite set of gradient vectors α_n. Gradient updates are obtained together with value updates around a prototype belief set, B. The updated vector $\alpha_{n+1}^{\{b\}}$ is retained in V_{n+1} only if it improves the value at point b as compared with V_n. Otherwise, $\alpha_n^{\{b\}}$ is admitted. When value backups are obtained using MILP, the gradient can be easily calculated using the maximizing action $a(\alpha_{n+1}^{\{b\}})$. It is also possible to obtain the gradient vector directly from the MILP solution (but this is not considered here).

3.3.2. Policy determination

Having obtained an estimate of optimal value function that satisfies a desired convergence criterion ε, the ε-optimal policy can be implemented by adopting one of the following on-line computational methods:

(i) Maximizing action associated with each gradient vector – During a value backup for belief point b, a gradient vector $\alpha_{n+1}^{\{b\}}$ is obtained that has a maximizing action $a(\alpha_{n+1}^{\{b\}})$ (22) associated with it. The corresponding maximizing vectors can be cached with each gradient vector. The ε-optimal policy π^ε corresponding to the ε-optimal value function V^ε is then given by (23) (shown for a belief point b).

$$\alpha_\varepsilon^{\{b\}} = arg\max_i \sum_s b(s)\alpha^i_{|V^\varepsilon|}(s) \tag{22}$$

$$\pi^\varepsilon(b) = a(\alpha_\varepsilon^{\{b\}}) \tag{23}$$

However, it may be expensive to hold the entire action descriptions, for each gradient vector, when the dimension of actions is large.

(ii) Solving on-line one step look ahead optimization – The ε-optimal policy $\pi^{\varepsilon LA}(b)$ for a belief point b can also be obtained in real time by running the value backup operation at each decision time. This is shown in (24), together with (3–5).

$$\pi^{\varepsilon LA}(b) = arg\max_{a\in A}\left\{\sum_{s\in S} r_a(s)b(s) + \gamma\sum_{o\in O} p(o|b,a)V^\varepsilon(b^{a,o})\right\} \tag{24}$$

Generally speaking, the first approach is computationally more favorable but may have high memory requirements if the dimension of the actions is large. It can be shown that policy $\pi^{\varepsilon LA}$ is at least as good as π^ε in an average sense.

3.4. Problem size versus computational complexity

In traditional enumeration based method, the problem size scales exponentially with size of state and action space due to the combinatorial structure of their interactions.

On the other hand, the size of the mixed integer model in Fig. 3, i.e., the number of variables and constraints, is dependent on $|O|$ and $|V_n|$. If model parameters $r_a(s)$, $t_a(s,s')$, and $op_a(s',o)$ satisfy the conditions imposed in Section 3.2, the mixed integer program is a MILP, which poses relatively little computational challenge. However, 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, $|V_n|$ is governed by a combination of factors including size of state space, action space as well as structure of the optimal policy. While approaches to limit the size of $|V_n|$ are dependent on the problem structure, we show a more general and elegant way to resolve the problem of large sized observation spaces in the following section.

4. Value iteration around post decision belief state

4.1. 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 of 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 Fig. 4. 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. $\tilde{b}_t^a$ is referred to as *post decision belief state* at time t.

In the context of POMDPs, solution using this approach is possible when the observation probabilities do not depend on action a. As pointed out in Section 3.2, 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.

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 Fig. 4. Intuitively, starting with a post decision state b^a, $|O|$ possible next states $b^{a,o}\forall o$ are obtained. For each of the states $b^{a,o}$, a maximizing action a' would determine the next post-decision belief state $\tilde{b}_t^a$ instead of the conventional pre-decision belief state $b^{a,o}$.

Let us consider the option of computing the value function in terms of post-decision belief state $b^{a',o}$ instead. First we must check if the structure of V^a is PWLC, a highly desirable property. To this end we first note that (Powell, 2007)

$$V^a(\tilde{b}^a) = \sum_o p(o|\tilde{b}^a, a)V^{pre}(b^{ao}) \tag{25}$$

and then use Lemma 1 to show that a positive weighted sum of convex functions is convex. Since $p(o|\tilde{b}^a) \geq 0$, the result holds.

Lemma 1. For two convex functions f and f' and scalar $\alpha \geq 0$

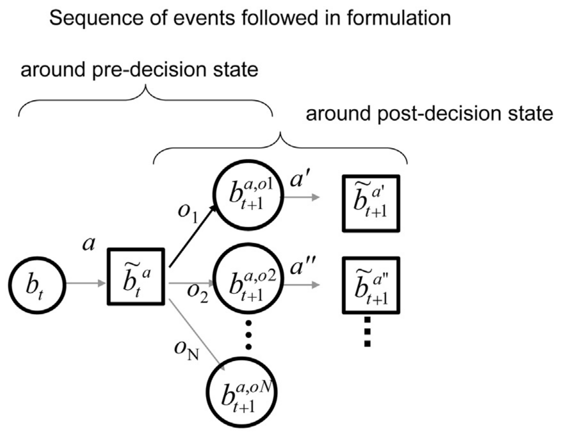

Fig. 4. A schematic of pre-decision state to post-decision state and again to pre-decision state.

(1) α, f is convex
(2) $f + f'$ is convex

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.

4.2. 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 (Matthijs & Vlassis, 2005). The value iteration step for the post decision state variable is given by (26) through (30). In (26), 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 $|O|$. Substituting (27) through (26) in (26)

$$V^a_{n+1}(\tilde{b}^a) = \sum_o p(o|\tilde{b}^a, a) \max_{a'} \left\{ \sum_s r(s, a') b^{ao}(s) + \gamma V^a_n(\tilde{b}^{a'}(s')) \right\} \tag{26}$$

$$b^{ao}(s) = \frac{\tilde{b}^a(s) p(o|s, a)}{\sum_s \tilde{b}^a(s) p(o|s, a)} \tag{27}$$

$$\tilde{b}^{a'}(s') = \sum_s b^{ao}(s) p(s'|s, a') \tag{28}$$

$$V^a_n(\tilde{b}^{a'}(s')) = \max_i \sum_{s'} \tilde{b}^{a'}(s') \alpha^i_n(s') \tag{29}$$

$$p(o|\tilde{b}^a, a) = \sum_s p(o|s, a) \tilde{b}^a(s) \tag{30}$$

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

Fig. 5. The MILP for determination of the maximizing action for value update for a post-decision belief state.

Introducing the notation

$$G^{a'o}(s) = T^{a'o}(s) + \gamma \arg\max_{\{g^i_{a'o}\}_i} < \tilde{b}^a, g^i_{a'o} > \tag{37}$$

and the identity (35), we finally obtain

$$V^a_{n+1}(\tilde{b}^a) = \sum_o \left\{ \tilde{b}^a . \arg\max_{\{G^{a'o}\}_{a'}} < \tilde{b}^a, G^{a'o} > \right\} \tag{38}$$

and

$$\alpha^{\{\tilde{b}^a\}}_{n+1} = \sum_o \arg\max_{\{G^{a'o}\}_{a'}} < \tilde{b}^a, G^{a'o} > \tag{39}$$

(38) and (39) give the value backup and gradient vector backup for the post decision belief state variable, respectively.

On lines similar to Section 3.1, the mathematical program for (36) for a given observation and belief state $\tilde{b}^a$ is shown in Fig. 5. 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 $o\_size \leq |O|$, where o_size is the number of observations for which $p(o|\tilde{b}^a) > 0$.

$$V^a_n(\tilde{b}^a) = \sum_o p(o|\tilde{b}^a, a) \max_{a'} \left\{ \frac{\sum_s r(s, a') p(o|s, a) \tilde{b}^a(s)}{\sum_s p(o|s, a) \tilde{b}^a(s)} + \gamma \max_i \sum_{s'} \sum_s \frac{p(o|s, a) p(s'|s, a') \tilde{b}^a(s) \alpha^i_n(s')}{\sum_s p(o|s, a) \tilde{b}^a(s)} \right\} \tag{31}$$

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

$$V^a_{n+1}(\tilde{b}^a) = \sum_o \max_{a'} \left\{ \sum_s r(s, a') p(o|s, a) \tilde{b}^a(s) + \gamma \max_i \sum_{s'} \sum_s p(o|s, a) p(s'|s, a') \tilde{b}^a(s) \alpha^i_n(s') \right\} \tag{32}$$

Using the notation

$$g^i_{a'o}(s) = \sum_{s'} p(o|s, a) p(s'|s, a') \alpha^i_n(s') \tag{33}$$

$$T^{a'o}(s) = r(s, a') p(o|s, a), \tag{34}$$

and the identity

$$\max_{\{y_j\}_j} x.y_j = x. \arg\max_{\{y_j\}_j} x.y_j \tag{35}$$

we can write (32) as

$$V^a_{n+1}(\tilde{b}^a) = \sum_o \max_{a'} \left\{ \sum_s T^{a'o}(s) \tilde{b}^a(s) + \gamma \tilde{b}^a . \arg\max_{\{g^i_{a'o}\}_i} < \tilde{b}^a, g^i_{a'o} > \right\} \tag{36}$$

4.2.1. 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) by storing and retrieving the maximizing action(s) associated with each gradient vector and (ii) by solving one step look-ahead optimization. In the new formulation, 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 a 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 optimization approach, on the other hand, the MILP needs to be solved for the belief state pertaining to the *current observation only*. This is computationally much less expensive than the look-ahead design for solution around pre-decision belief state, and therefore a major advantage.

4.3. Comparison with value updates around pre-decision belief states

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

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

(ii) Policy determination in real time – As discussed in the previous section and Section 3.3, the pre-formulation allows for storing optimal action with each gradient vector that characterizes the ε-optimal value function. For the post-formulation, 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 optimization for action determination should be considerably faster for the post-formulation due to the lower complexity of the associated MILP.

(iii) Handling very large or continuous observation spaces – While the pre-formulation is limited to small observation spaces, the MILP technique for value updates based on the 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 the multiple MILPs.

Having derived the necessary models and equations, the following section is devoted to two illustrative examples. The examples are aimed at demonstration of the technique and analysis of solution times and solution quality as a function of problem size.

5. Illustrative examples

5.1. POMDP with continuous actions

In order to illustrate the concept of using mathematical programming for value backups, a simple, admittedly artificial, 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 the more desirable state than 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 increases the likelihood of it returning to state s_1 from state s_2. These actions can be thought of as degrees of preventive and corrective actions taken to ensure the equipment is in state s_1, e.g., cleaning, lubrication, etc. The tasks are scaled to obtain the bounds of 0 and 1.

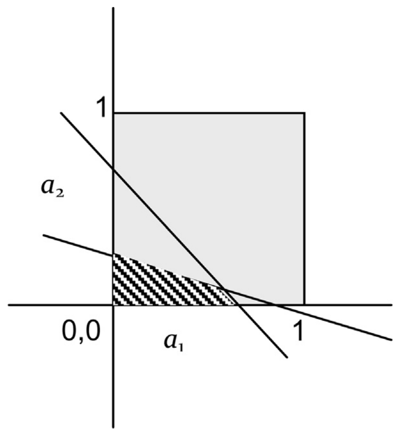

Fig. 6. Feasible region for the continuous two-action problem (when $b_1 = b_2 = 0.5$).

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

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

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 $0 < \beta < 1$. The resulting observation and reward matrices are as shown below:

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

Finally, there are system and budget constraints of the form shown in (40) and (41), 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.

$$A_1a_1 + A_2a_2 + A_3 \le 0 \tag{40}$$

$$B_{11}b_1a_1 + B_{12}b_1a_2 + B_{21}b_2a_1 + B_{22}b_2a_2 + B_3 \le 0 \tag{41}$$

The feasible region of the MILP, in the absence of constraints (36) and (37), is given by the shaded region with hashed line in Fig. 6. Therefore, in the absence of constraints (36) and (37), the solution of the above problem would lie on the extreme points, i.e., optimal values of both a_1 and a_2 are either 0 or 1. In such a scenario, the action space is practically discrete and substantially smaller.

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	0.05	0.3	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

The MILP model for the value backups for above example is shown by model M2. This pertains to value iteration around pre-decision state variable.

Model M2

$$\max_a \left\{ \sum_s b(s) r(s) + \gamma \sum_o v(o) \right\}$$

$s.t.$

$$r(s) = R(s) - C(s, s')a(s')$$

$$t = T_a(a)$$

$$Aa \le 0$$

$$Bba \le 0$$

$$v(o) \ge btp(o)\alpha$$

The converged value function and policy for the two approaches are shown in Fig. 7. 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. When the probability of being in state s_1 is sufficiently high, action a_1 is executed whose value depends on the constraints and the objective value. The policy for the two action problem is rather simple and intuitive. When the probability of being in state s_1 falls below a certain threshold (different for both approaches), action a_2 is performed so that it satisfies the constraints listed in (40) and (41). As seen in Fig. 7, the value function for the enumeration based solution is lower. The performance and solution times of MILP and enumeration based methods (as a function of number of PERSEUS iterations) are shown in Fig. 8(a) and (d), respectively.

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. The action corresponding to each gradient vector is also shown in Fig. 7. As seen, the actions pertaining to the best solution obtained by enumeration (of action space) is limited by the size of the action space grid. For the same reason, the MILP based method appears to be converging faster in terms of number of iterations as seen in Fig. 8(d).

To study how the solution time scales with the problem size and to 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 (40) and (41). The corresponding results are reported in Table 2. It is observed that the performance gap widens as the problem size grows. The phenomenon is seen in Fig. 7(d) through (f) as well. Additionally, unlike the case with the two-action system, the convergence times are of the same order of magnitude for three-action system and the convergence time is an order of magnitude higher for the enumeration based technique as the problem size grows to four actions. As the problem size grows, the number of actions grows exponentially in the case of the enumeration based method. Also due to coarse-grid sampling of the action space, the technique tends to maintain many more gradient vectors to closely approximate the value function. The two factors, i.e., $|A|$ and $|V_n|$, contribute to the rapid increase in solution times as the problem size increases. For the MILP based method, the number of actions as well as the number of gradient vectors increase moderately with problem size. The solution times therefore grow marginally as the problem size grows. This can be concluded from the fact that the plots corresponding to the MILP based method are almost identical, i.e., they converge after about 20 iterations.

The shape of the solution time versus number of iterations plot reveals information about the dependence of solution time per iteration on $|V_n|$ where n is the iteration counter. It is observed that, for both the MILP and enumeration based methods, the number of iteration versus solution time has three distinct regions:

(i) For the first few iterations, the cumulative solution time is linear with a very small slope.
(ii) After this, the solution time grows exponentially with iteration count.
(iii) Finally, it becomes linear again with a large slope.

This is because the value function is initialized with a single gradient vector and the value function is set to the lowest possible reward. Due to this reason, a single update improves the value function for the entire sampled belief set. Consequently, only a single gradient vector comprises the value function for the first few iterations leading to same solution time per iteration. Beyond this point, the size of the value function, i.e., the number of gradient vectors increases with iterations as the shape of the value function begins to take form. This causes the solution time to increase per iteration, as more gradient vectors are added. Finally, the value function converges and no new gradient vectors are added. At this point, the solution time again becomes constant with iterations. The time per iteration is much higher as compared to that for the initial iterations.

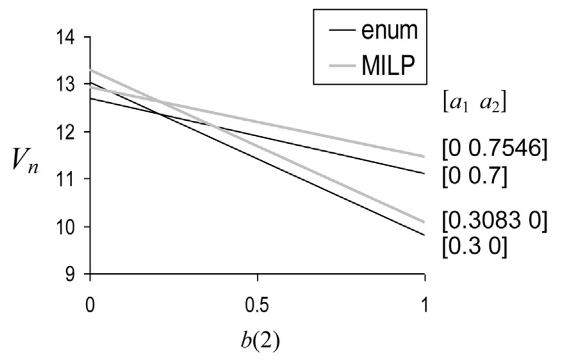

Fig. 7. Value function and policy comparison for two action system.

5.2. Flow network problem

In this subsection, we apply the proposed MILP based value backup procedure to the flow network example introduced in Section 2.

5.2.1. MILP based value backups

The MILP formulation must account for the following:

(i) The constraints to exclude infeasible actions.
(ii) The equations to develop state transition matrix map_2 as a function of actions.

Whereas the constraints to exclude infeasible actions (described earlier in this section) are relatively easily developed, the second

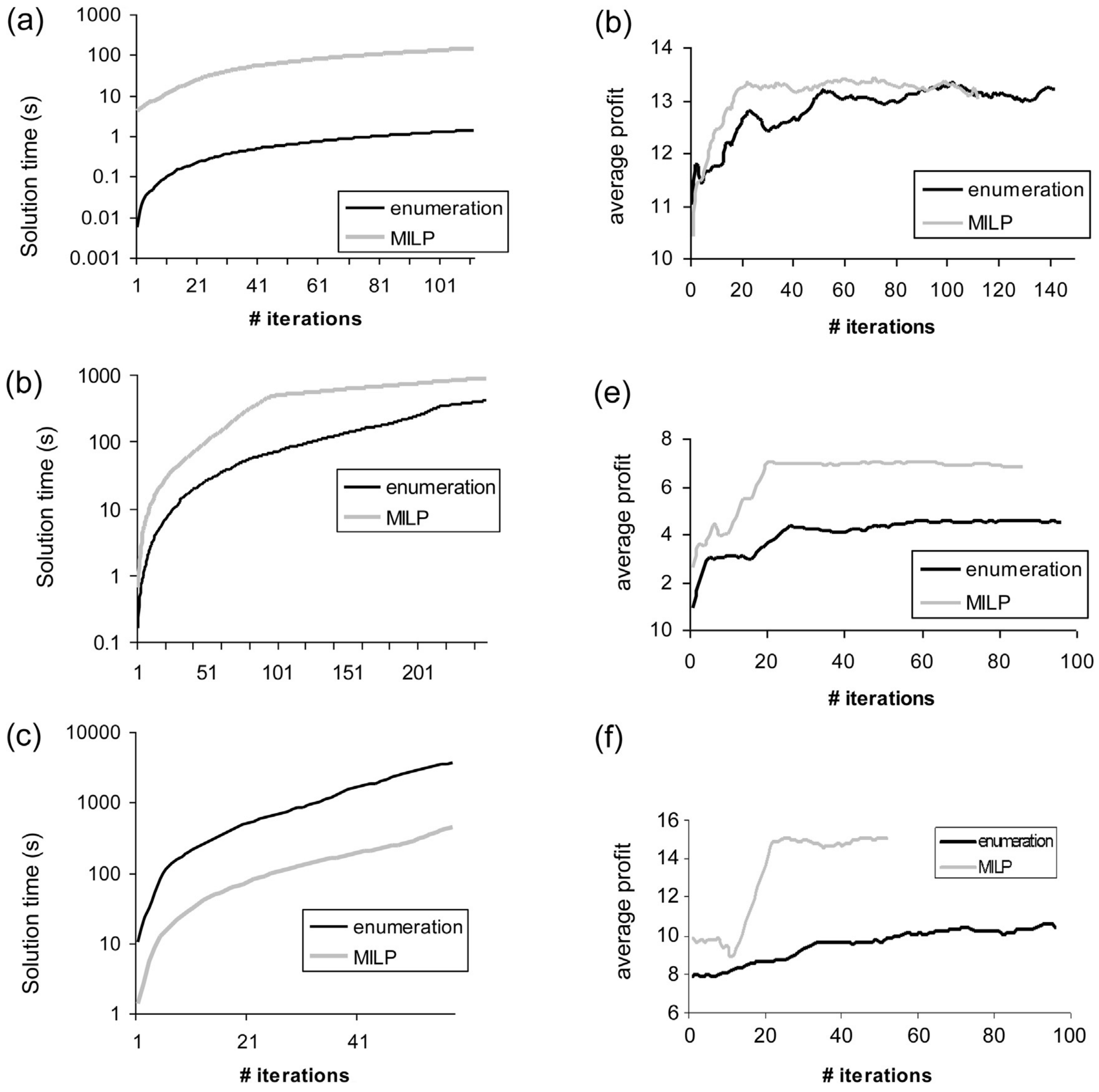

Fig. 8. The comparison of convergence times and performance for the problem with continuous actions. Number of iterations versus solution time for the problem with (a) two continuous actions, (b) three continuous actions, (c) four continuous actions. Average profit as a function of number of iterations for the problem with (d) two continuous actions, (e) three continuous actions, (f) four continuous actions.

set of constraints and equations to generate map_2 is a more difficult task. The MILP backup for a post-decision belief state b^a is developed by using its predecessor pre-decision belief state, for each of the possible observations $b^{a,o} \forall o$. For a given o, let $b = b^{a,o}$ and let $b^{a'}$ denote the next post-decision belief state. The following sequence of events is followed:

(i) At time $t = 0$, the repair decision is implemented. The system transitions from state s to an intermediate state s_1.
(ii) The node connectivity is determined and the appropriate reward generated.
(iii) Then the flow decisions are taken and the state of the system at the next time is s_2. This is because it is assumed that a contaminated node corrupts a downstream node at the next time.
(iv) Finally, the contamination probabilities are realized and the cycle is repeated.

The MILP model for this process is shown in Fig. 9. M3.1 through M3.3 determine the value function for $b^{a'}$, M3.4 and M3.5 model the input–output constraints described before and M3.6, M3.7, are the constraints that ensure that there is no cyclical flow between any two, three,. . .,N nodes. The equations are only shown for two

Table 2
Results for the problems with continuous actions.

Problem	$\|S\|$	$\|A\|$	$\|O\|$	Performance		$\|V_n\|$		Convergence time	
				MILP	Enum	MILP	Enum	MILP	Enum
2 Actions	2	121	2	13.28 ± 1.2	13.08 ± 1.5	6	7	16.02	0.62
3 Actions	3	1331	3	16.71 ± 2.1	14.67 ± 3.2	15	11	46.76	29.45
4 Actions	4	14641	4	15.02 ± 1.9	10.27 ± 3.6	16	159	108.22	1180.25

$$\max \sum_{i,l} b(l)\{C_P w(i) z(l,i) - C_{repair}(i) a_{repair}(i)\} - \sum_{i,j} C_{flow}(i,j) a_{flow}(i,j) - 100 \sum_{l,l1,i} t(l,l1,i) + 0.9v$$

$$s.t.$$

$$v \geq \sum_{l} \widetilde{b}^{a'}(l) \alpha(m,l) \qquad \forall m \qquad \text{M3.1}$$

$$v \leq \sum_{l} \widetilde{b}^{a'}(l) \alpha(m,l) + M2(1 - y(m)) \qquad \forall m \qquad \text{M3.2}$$

$$\sum_{m} y(m) = 1 \qquad \text{M3.3}$$

$$\sum_{j} a_{flow}(j,i) - a_{flow}(i,i) \geq a_{flow}(i,i) \qquad i = 2,3,..N \qquad \text{M3.4}$$

$$\sum_{j} a_{flow}(j,i) - a_{flow}(i,i) \geq a_{flow}(i,k) \qquad i = 2,3,..N, \forall k \qquad \text{M3.5}$$

$$a_{flow}(i,j) + a_{flow}(j,i) \leq 1 \qquad \forall i \neq j \qquad \text{M3.6}$$

$$a_{flow}(i,j) + a_{flow}(j,k) + a_{flow}(k,i) \leq 2 \qquad \forall i \neq j \neq k \qquad \text{M3.7}$$

$$....$$

$$z(l,i) \leq a_{flow}(i,i) \qquad \forall l,i \qquad \text{M3.8}$$

$$z(l,i) \leq 1 - s1(l,i) \qquad \forall l,i \qquad \text{M3.9}$$

$$s1(l,i) \geq s(l,i) - a_{repair}(i) \qquad \forall l,i \qquad \text{M3.10}$$

$$s2(l,i) \geq s1(l,i) \qquad \forall l,i \qquad \text{M3.11}$$

$$1 - s2(l,k) \leq 1 - s1(l,i) + 1 - a_{flow}(i,k) \qquad \forall l, i \neq k \qquad \text{M3.12}$$

$$\sum_{i} \tau(l,l1,i) \geq s2(l,i) - s(l1,i) \qquad \forall l,l1 \qquad \text{M3.13}$$

$$\sum_{i} \tau(l,l1,i) \geq -1\{s2(l,i) - s(l1,i)\} \qquad \forall l,l1 \qquad \text{M3.14}$$

$$map_2(l,l1) \geq 1 - \sum_{i} \tau(l,l1,i) \qquad \forall l,l1 \qquad \text{M3.15}$$

$$\sum_{l1} map_2(l,l1) = 1 \qquad \forall l \qquad \text{M3.16}$$

$$\widetilde{b}^{a'}(l2) = \sum_{l,l1} b(l) map_2(l,l1) map_1(l1,l2) \qquad \forall l \qquad \text{M3.17}$$

Fig. 9. Model M3: MILP model for the value backup for network flow POMDP.

Table 3
Parameter values for the network flow problems.

Parameter description	Symbol	Value	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
Origination probability for six-nodes problem	p_i		0	0.04	0.025	0.015	0.01	0.01
For five-nodes problem	p_i		0	0.04	0.025	0.015	0.02	
For four-nodes problem	p_i		0	0.04	0.025	0.035		
Weighting factor	w_i		–	6	4	7	6	8
Repair cost per node	C_{repair}	30						
Reward for each active node	C_P	1						

$$\begin{bmatrix} 0000 \\ 0001 \\ 0010 \\ 0011 \\ 0100 \\ 0101 \\ 0110 \\ 0111 \end{bmatrix} \quad \xrightarrow[{[0\,0\,0\,1]}]{a_{repair}=} \quad \overset{l=4}{\begin{bmatrix} 0010 \\ \ldots \end{bmatrix}} \quad \xrightarrow[=1]{a_{flow}(3,2)} \quad \overset{l'=7}{\begin{bmatrix} 0\,1\,1\,0 \\ .. \end{bmatrix}}$$

Fig. 10. Illustration for mapping state l to l'.

$$C_{flow} = \begin{bmatrix} 0 & 0.1 & 0.2 & 0.5 & 1 & 0.5 \\ 10 & 0.1 & 0.1 & 0.5 & 0.8 & 0.3 \\ 10 & 0.1 & 0 & 0.1 & 0.1 & 0.1 \\ 10 & 0.5 & 0.1 & 0 & 0.5 & 1 \\ 10 & 0.8 & 0.5 & 0.1 & 0 & 0.8 \\ 10 & 0.7 & 1 & 0.2 & 0.4 & 0.1 \end{bmatrix}$$

Fig. 11. Cost parameter for the network flow problem with four, five and six nodes.

and three node combinations. This is similar to saying that a node cannot supply and receive material from another node at the same time.

A variable $z(l,i)$ is introduced to denote the status of connectivity of node i in state $s_1(l,i)$ where $l = 1,2,.|S|$. $z(l,i)$ is determined by M3.8 and M3.9. The effects of repair and flow actions on state transitions are determined by M3.10 through M3.12. Having found a state s_2 (l,i) corresponding to each state s_1 (l,i) in the state space, the key is to develop a map for the transition in order to determine the belief state $b^{a'}$. This is determined by obtaining the index of s_2 within the state space with the help of variables $\tau.\tau(l,l',i) = 0 \quad \forall i$ when $s_2(l,i)$ is equal to index l' of the state space. This implies that state designated by l transitions to l'. Correspondingly, $map_2(l,l')$ is set to 1. This is achieved by equations M3.13 through M3.16. The concept is further illustrated by Fig. 10. Once map_2 is determined, $b^{a'}$ is calculated using M3.17.

6. Results and discussion

The network flow problem described above is solved for four, five and six nodes. In all three cases, nodes η_2 and η_3 have measurement sensors. The parameter values are shown in Table 3 and Fig. 11. The solution for the three problem instances is shown in Table 4. The performances are comparable for the two methods. This is because the exact same problems are being solved in this case as opposed to using a discretized action space in the previous example. The solution times on the other hand follow a similar trend as the previous example in that it increases very rapidly with size for the enumeration based solution methods. Since $|V_n|$ is similar for the two solution approaches (in all three cases), the difference in solution times is almost entirely attributed to the exponentially increasing action space size and the combinatorial nature of the enumeration based method. The behavior of the solution times is better understood by Fig. 12(a–c). As noted before in the previous example, both for the MILP and enumeration based methods, the number of iteration versus solution time has three distinct regions, for the reasons stated earlier.

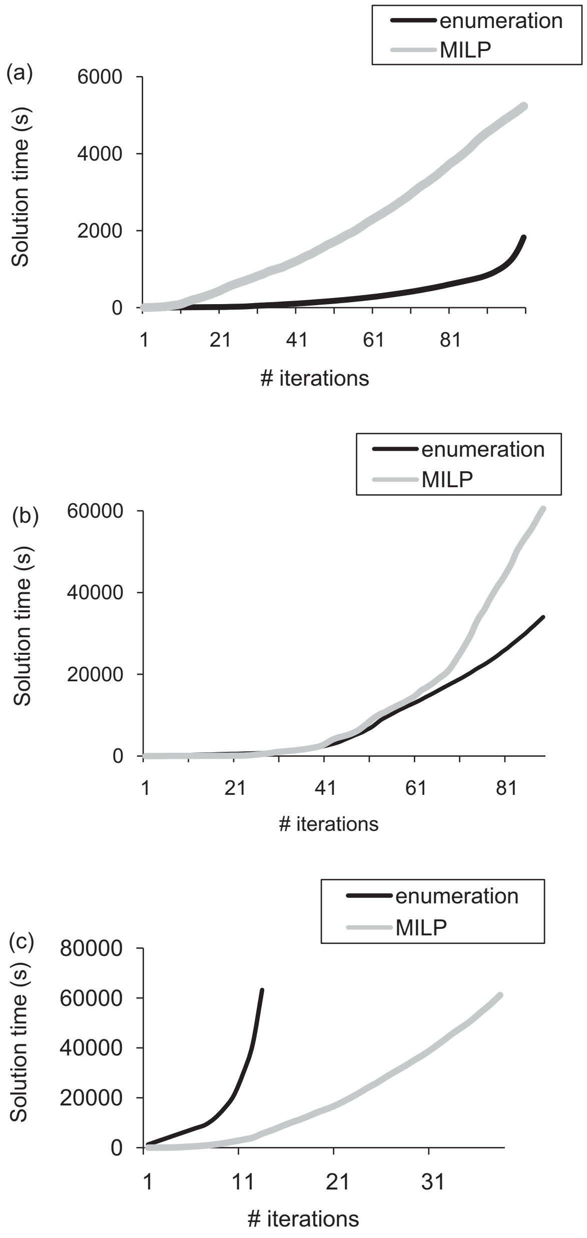

Fig. 12. Comparison between solution times of the MILP based backups and the enumeration based backups for the network flow problem with (a) four nodes, (b) five nodes and (c) six nodes.

Table 4
Problem size and results for the network flow problems.

Problem	$\|S\|$	$\|A\|$	$\|O\|$	Performance	Performance	$\|V_n\|$		Convergence time	
				MILP	Enum	MILP	Enum	MILP	Enum
Network flow									
4 Nodes	8	2336	4	179.54 ± 13.56	181.22 ± 12.51	22	39	174	97
5 Nodes	16	15,296	4	225.91 ± 21.45	223.96 ± 25.76	42	43	1168	3132
6 Nodes	32	63,744	4	292.51 ± 10.72	292.67 ± 9.94	73	67	5641	63,279

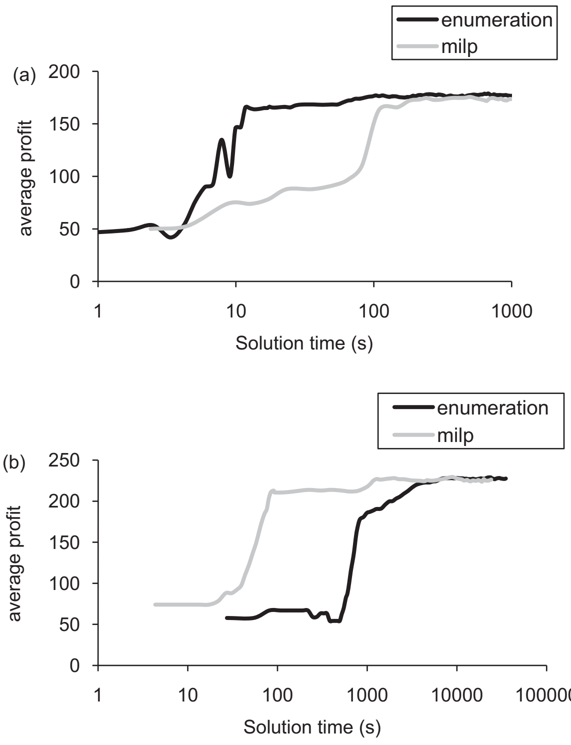

Fig. 13. Comparison of convergence and performance of the MILP and enumeration approaches for the network flow problem with (a) four nodes, (b) five nodes.

Finally, the convergence of value function is depicted in Fig. 13(a) and (b) for the four and five node problems, respectively. The average performance is plotted as a function of solution time. For the four node network flow problem, the value function converges faster for the enumeration based method. However, the phenomenon is reversed starting with the five node problem. As seen from Table 2, the relative performance of the MILP based method only gets better with increase in problem size.

7. Conclusions

In this work, MILP models were developed to obtain value backups for POMDPs with very large or continuous action spaces. The traditional way of doing this is by using either a homogeneous or heterogeneous grid or sampling methods. By using MILP based methods, exact calculation of the max operator for each value backup can be ensured, thereby preserving the solution quality.

The MILP is amenable to large computation times when the size of the observation space is large. For this reason, an alternative MILP model is developed for solution of POMDP around post decision belief state. This removes the dependence of MILP solution time on the size of the observation space. An example with continuous action space is presented to illustrate the solution method. It is observed that the MILP based method produces higher quality result, when a homogenous grid is used (for the action space) for the enumeration based method. The solution gap increases with problem size.

Additionally, the solution time scales linearly with problem size for the MILP based method, while it grows exponentially in the case of the enumeration based method. This is attributed to the exponential increase in the size of action space due to combinatorial reasons. Consequently, the MILP quickly surpasses the enumeration based method in terms of lower solution times and better solution quality as the problem size is increased. This makes the MILP based solution very well suited for POMDPs with high dimensional discrete or continuous action/observation spaces.

Acknowledgment

This research was supported by the Advanced Biomass R&D Center (ABC) of Global Frontier Project funded by the Ministry of Education, Science and Technology (ABC-0031354), the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (20110006839), and the WCU (World Class University) program through the NRF funded by the Ministry of Education, Science and Technology (R32-2008-000-10142-0).

References

Cassandra, A. R. (1998). *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes. A Survey of POMDP Applications* (pp. 17–24).
Ng, A. Y., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Uncertainty in Artificial Intelligence*.
Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, *26*(2), 282–304.
Baxter, J., & Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.
Hoey, J., & Poupart, P. (2005). Solving POMDPs with continuous or large discrete observation spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
Lee, J., & Unnikrishnan, S. (1998). Planning quality inspection operations in multistage manufacturing systems with inspection errors. *International Journal of Production Research*, *38*(1).
Berry, J. W. (2005). Sensor placement in municipal water networks. *Journal of Water Resources Planning and Management*, *131*(3), 237–243.
Bellman, R. (1957). *Dynamic programming*. Princeton University Press.
Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, *99*(1), 21–71.
Matthijs, T. J. S., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 26.
Rico-Ramirez, V., Frausto-Hernandez, S., Diwekar, U. M., & Hernandez-Castro, S. (2007). Water networks security: A two-stage mixed-integer stochastic program for sensor placement under uncertainty. *Computers & Chemical Engineering*, *31*(5/6), 565–573.
Powell, W. B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality*. Wiley-Interscience.