Using XADDs and Symbolic Dynamic Programming to solve continuous POMDPs

Justin Dieter
jdieter@stanford.edu

Abstract

Point Based Value Iteration (PBVI) has been extremely successful in producing solutions to Partially Observable Markov Decision Processes (POMDPs) with discrete state and observation spaces. With sampling or discretization, it is possible to extend this to continuous action and observation spaces but it is desirable to produce exact backups for belief points in POMDPs with continuous state and observation. Recently, PBVI has been extended using Symbolic Dynamic Programming to provide exact point based backups for very special cases of continuous state and observation POMDPs. This paper gives a survey of this method and analyzes its effectiveness.

1 Introduction

PBVI has proven to be successful at approximating solutions for many POMDPs using point based backups [1]. With sampling this can even be extended to continuous observation and state spaces. However, often POMDPs arise with extra structure that can be exploited with a symbolic method to produce exact backups [3]. This algorithm is called Symbolic Dynamic Programming (SDP). An open source implementation of this algorithm in Julia is included alongside this paper and can be accessed at `https://github.com/jdietz31/SDPPOMDP`. This paper gives a survey of the workings of the algorithm as well as an analysis of its performance on a sample POMDP.

2 Exetended Algebraic Decision Diagrams (XADDs)

Extended Algebraic Decision Diagrams (XADDs) give an excellent method for representing piecewise functions, as well as performing operations on them [2]. An XADD is a tree. There are terminal nodes that contain an expression representing a real number in terms of the given input variables. There are also, earlier nodes that contain expressions that are either equalities or inequalities and point to a further nodes decided on whether the given expression is true or false. An XADD can represent any piecewise function by starting at a root node and following the decision nodes until a terminal node is reached. While, an arbitrary piecewise continuous function can theoretically be represented with this structure, for the purpose of the specific types of POMDPs being considered, XADDs containing only linear inequalities and linear terminal nodes are sufficient. For example the piecewise continuous function,

$$f(x,y) = \begin{cases} 2x + 2y & \text{if } x + y \geq 0 \\ -2x - 2y & \text{if } x + y < 0 \text{ and } y < 0 \\ 2 & \text{otherwise} \end{cases} \tag{1}$$

is compactly represented by the XADD in figure (1).

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

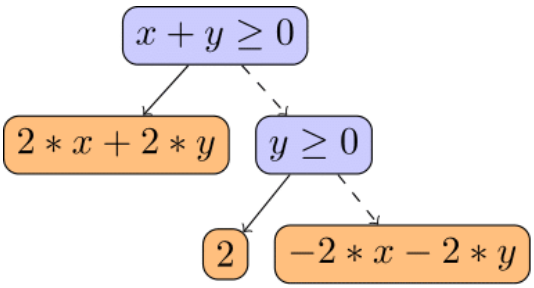

Figure 1: An XADD version of the piecewise function in equation (1)

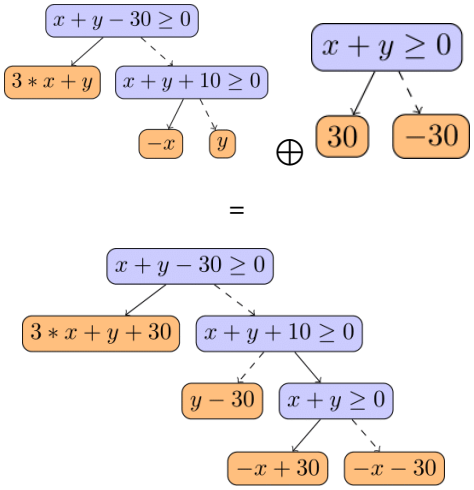

Figure 2: An example of XADD addition (multiplication, max, and min work similarly)

2.1 Reducing XADDS

As XADDs are manipulated with this algorithm, they will grow to be large and computationally challenging to represent. Fortunately, it is tractable to reduce an XADD to its simplest form, pruning any redundant or unreachable nodes. Pruning an XADD involves doing a breadth-first search starting at the root node. A linear program can be grown by adding the decision nodes in the path being searched as constraints. If it becomes unsolvable then the path is unreachable and should be pruned. Removing redundant nodes is slightly more complicated. It involves searching through each decision node and seeing if one of the subtrees it produces contains the other child node. If so, it is redundant and is pruned. There are however, several optimizations in the algorithm to make it tractable. [2]

2.2 XADD Operations

The critical insight of this algorithm is that all the operations that would be desirable to compute of piecewise functions can easily be computed with the XADD representation.

2.2.1 Binary Operations

Addition or multiplication ($\oplus$, $\otimes$) of XADDs is done by recursively building a new tree starting from the root nodes to consider every possible decision from both trees and respectively multiplying or adding the terminal nodes when they are reached. Figure (2) gives an example of this operation. The max and min operations work similarly but often create a bad ordering of decisions and an extra reorder operation is necessary to keep a valid tree. [2]

Γ_t=BACKUP(B, Γ_{t-1})	1
For each action $a \in A$	2
For each observation $z \in Z$	3
For each solution vector $\alpha_i \in \Gamma_{t-1}$	4
$\alpha_i^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha_i(s'), \forall s \in S$	5
End	6
$\Gamma_t^{a,z} = \cup_i \alpha_i^{a,z}$	7
End	8
End	9
$\Gamma_t = \emptyset$	10
For each belief point $b \in B$	11
$\alpha_b = \text{argmax}_{a \in A} \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} \max_{\alpha \in \Gamma_t^{a,z}} \left[\sum_{s \in S} \alpha(s) b(s) \right] \right]$	12
If($\alpha_b \notin \Gamma_t$)	13
$\Gamma_t = \Gamma_t \cup \alpha_b$	14
End	15
Return Γ_t	16

Figure 3: Outline of the standard PBVI algorithm

2.2.2 Unary Operations

Unary operations include scalar multiplication, substitution and integration. Scalar multiplication is implemented by simply applying it to the terminal nodes. Substitution is the process of substituting a variable for a different expression. The substitution of a variable y for expression x in a function f is denoted by $f\{y/x\}$. This is almost as easy as just replacing the expressions in every node other than that the order of the decisions can become incorrect. However, the same reorder operation used for the max and min operations will resolve this issue. Integration would seem to be a very difficult problem, but it is only necessary to integrate over very specific functions containing a Dirac delta. Thus the integration can be replaced with a simple substitution operation as shown below:

$$\int_y \delta[y - g(\overrightarrow{x})] f = f\{y/g(\overrightarrow{x})\} \tag{2}$$

3 Point Based Value Iteration (PBVI)

Point Base Value Iteration provides an effective technique for producing approximate solutions to POMDPs. [1] It is the algorithm we will extend to produce the symbolic algorithm we desire. As outlined in figure (3), PBVI starts with a set of belief points and creates α-vectors to approximate their value. At each iteration, it performs a backup to produce slightly more accurate α-vectors. However, it is approximate since the α-vectors are only calculated over the belief points provided. Also, in order to be extended to continuous state and observation spaces, sampling techniques must be used increasing the error of approximation.

4 Exact Value Iteration for POMDPs

The kind of approach used in PBVI can be extended to solve finite POMDPs exactly despite being computationally impractical for large state and action spaces. If α-vectors are collected for all potential belief points, not just the optimal ones it can be solved exactly. However, when using this approach line 12 in the PBVI algorithm becomes much more complicated since a large set of α-vectors need to be maintained. At each timestep h, intermediate sets of α-vectors, Γ^h, need to be maintained to determine the updates to the α vectors. These are created using the following formulas

for all $a \in A$, $o \in \mathcal{O}$, and $\alpha_j \in \Gamma^{h-1}$

$$g^h_{a,o,j}(s) = \sum_{s' \in S} T(s,a,s')O(s',a,o)\alpha_j(s') \tag{3}$$

$$\Gamma^h_a = R(s,a) + \gamma \boxplus_{o \in \mathcal{O}} [\{g^h_{a,o,j}(s)\}_j] \tag{4}$$

$$\Gamma^h = \bigcup_a \Gamma^h_a \tag{5}$$

where $\boxplus$ represents the pairwise cross-sum ($\boxplus_i[S_i] = S_1 \boxplus S_2 \boxplus ...$ and $P \boxplus Q = \{p+q | p \in P, q \in Q\}$). It is easy to see how with large state or observation spaces and even with some optimizations, this algorithm will become computationally infeasible. However, elements of it will be necessary for the point based symbolic method.

5 Hybrid POMDPs (H-POMDPs)

The symbolic method of PBVI will prove to be very useful for solving a specific kind of POMDP, the H-POMDP. The H-POMDP is a tuple $(S, A, \mathcal{O}, T, R, O, \gamma, h)$. S denotes the state space and contains vectors of the form $(\mathbf{d}_s, \mathbf{x}_s) = (d_{s_1}, d_{s_2}, ..., d_{s_n}, x_{s_1}, x_{s_2}, ..., x_{s_m})$ where each d_{s_i} is a boolean value and each x_{s_i} is a real value. Similarly the observation space $\mathcal{O}$ is assumed to be of the form $(\mathbf{d}_o, \mathbf{x}_o) = (d_{o_1}, d_{o_2}, ..., d_{o_p}, x_{o_1}, x_{o_2}, ..., x_{o_q})$. The action space A is assumed to be discrete. Rewards in $R : S \times A \to \mathbb{R}$ must be given by a piecewise linear function that can be represented by an XADD. γ denotes the discount factor and h the timestep. Transition probabilities T and observation probabilities O are represented by a Bayesian network:

$$T : p(\mathbf{x}_{s'}, \mathbf{d}_{s'} | \mathbf{x}_s, \mathbf{d}_s, a) = \prod_{i=1}^{n} p(d_{s'_i} | \mathbf{x}_s, \mathbf{d}_s, a) \prod_{j=1}^{m} p(x_{s'_j} | \mathbf{d}_{s'}, \mathbf{x}_s, \mathbf{d}_s, a) \tag{6}$$

$$O : p(\mathbf{x}_o, \mathbf{d}_o | \mathbf{x}_{s'}, \mathbf{d}_{s'}, a) = \prod_{i=1}^{p} p(d_{o_i} | \mathbf{x}_{s'}, \mathbf{d}_{s'}, a) \prod_{j=1}^{q} p(x_{o_j} | \mathbf{x}_{s'}, \mathbf{d}_{s'}, a) \tag{7}$$

Probabilities over discrete observations and state transitions must be piecewise linear. Also, transitions and observations for continuous elements must be deterministic and piecewise linear. The probabilities must be represented by delta functions of piecewise linear functions. While this seems quite restrictive, there are a great deal of continuous problems that can be represented by this H-POMDP structure. Also, the piecewise linear requirement is not all that restrictive as arbitrary piecewise functions can be represented arbitrarily well with piecewise linear approximations.

6 Symbolic Dynamic Programming (SDP)

Symbolic Dynamic Programming utilizes some of the techniques of PBVI and exact value iteration to provide exact point based backups of belief points in H-POMDPs. The key insight is that if α-vectors are chosen to be represented as α-functions that are piecewise linear and represented by XADDs, all the operations of dynamic programming can be conveniently represented as XADD operations. Specifically, if the observation space is assumed to be discrete (continuous observations will be dealt with later), equation (3) becomes

$$g^h_{a,o,j}(\mathbf{x}_s, \mathbf{d}_s) = \int_{\mathbf{x}_{s'}} \bigoplus_{\mathbf{d}_{s'}} [p(o | \mathbf{x}_{s'}, \mathbf{d}_{s'}, a) \otimes (\bigotimes_{i=1}^{n} p(d_{s'_i} | \mathbf{x}_s, \mathbf{d}_s, a)) \otimes (\bigotimes_{j=1}^{m} p(x_{s'_j} | \mathbf{d}_{s'}, \mathbf{x}_s, \mathbf{d}_s, a)) \otimes \alpha_j(\mathbf{x}_{s'}, \mathbf{d}_{s'})] d\mathbf{x}_{s'} \tag{8}$$

which can all be represented by the earlier defined XADD operations. The only extra addition to the exact value iteration used here is that after every iteration, the alpha vectors are pruned to only

Algorithm 2: GenRelObs($\Gamma^{h-1}, a, \mathbf{b}_i$) $\longrightarrow \langle \mathcal{O}^h, p(\mathcal{O}^h|\mathbf{x}'_s,\mathbf{d}'_s, a)\rangle$

1 **begin**
2 **foreach** $\alpha_j(\mathbf{x}'_s,\mathbf{d}'_s) \in \Gamma^{h-1}$ *and* $a \in A$ **do**
3 *// Perform exact 1-step DP backup of α-functions at horizon $h-1$*
4 $\alpha_j^a(\mathbf{x}_s,\mathbf{d}_s,\mathbf{x}_o,\mathbf{d}_o) := \int_{\mathbf{x}'_s} \bigoplus_{\mathbf{d}'_s} p(\mathbf{x}_o,\mathbf{d}_o|\mathbf{x}'_s,\mathbf{d}'_s,a) \otimes p(\mathbf{x}'_s,\mathbf{d}'_s|\mathbf{x}_s,\mathbf{d}_s,a) \otimes \alpha_j(\mathbf{x}'_s,\mathbf{d}'_s)\, d\mathbf{x}'_s$
5 **foreach** $\alpha_j^a(\mathbf{x}_s,\mathbf{d}_s,\mathbf{x}_o,\mathbf{d}_o)$ **do**
6 *// Generate value of each α-vector at belief point $\mathbf{b}_i(\mathbf{x}_s,\mathbf{d}_s)$ as a function of observations*
7 $\delta_j^a(\mathbf{x}_o,\mathbf{d}_o) := \int_{\mathbf{x}_s} \bigoplus_{\mathbf{d}_s} \mathbf{b}_i(\mathbf{x}_s,\mathbf{d}_s) \otimes \alpha_j^a(\mathbf{x}_s,\mathbf{d}_s,\mathbf{x}_o,\mathbf{d}_o)\, d\mathbf{x}_s$
8 *// Using* casemax*, generate observation partitions relevant to each policy – see text for details*
9 $\mathcal{O}^h :=$ extract-partition-constraints[casemax$(\delta_1^{a_1}(\mathbf{x}_o,\mathbf{d}_o), \delta_1^{a_2}(\mathbf{x}_o,\mathbf{d}_o), \ldots, \delta_j^{a_r}(\mathbf{x}_o,\mathbf{d}_o))$]
10 **foreach** $o_k \in \mathcal{O}^h$ **do**
11 *// Let ϕ_{o_k} be the partition constraints for observation $o_k \in \mathcal{O}^h$*
12 $p(\mathcal{O}^h = o_k|\mathbf{x}'_s,\mathbf{d}'_s,a) := \int_{\mathbf{x}_o} \bigoplus_{\mathbf{d}_o} p(\mathbf{x}_o,\mathbf{d}_o|\mathbf{x}'_s,\mathbf{d}'_s,a)\mathbb{I}[\phi_{o_k}] d_{\mathbf{x}_o}$
13 **return** $\langle \mathcal{O}^h, p(\mathcal{O}^h|\mathbf{x}'_s,\mathbf{d}'_s,a)\rangle$
14 **end**

Figure 4: Outline of the algorithm to reduce to discrete observations.

contain the ones that are optimal at each belief point in a similar way to what PBVI does. This keeps the problem tractable but still ensures that exact backups are produces at least for the belief points inputed. For the given starting belief points, this algorithm provides exact values for the H-POMDP.

7 Dealing With Continuous Observations

While it is impossible to use this approach with arbitrary continuous observations, there is a trick that can be exploited in the structure of H-POMDPs. If a dynamic programming backup is used on the α-functions to get them in terms of observation, the only data that is needed from the observations is which alpha vector is optimal for a given observation. The continuous observation can then be switched with the discrete observation of which alpha vector is optimal and no data will be lost. Luckily, this can all be accomplished with the basic XADD operations. Figure (4) outlines the algorithm for extracting this discrete observation representation from the continuous observations. The new observations are extracted in lines 2 through 9 and the probabilities of the new observations are computed in lines 10 through 12.

8 Experiments

The algorithm is tested on a sample H-POMDP. The **1-D Power Grid** has a single continuous state variable, a single discrete action, and a single discrete observation. In this POMDP, a valve is manipulated with the goal of maintaining a safe temperature for a power grid. Having a closed valve is desirable, but the temperature of the system increases with a closed valve so it needs to be opened occasionally. The state transitions for the system are given by the following equation,

$$p(t'|t,a) = \delta\left[t'_s - \begin{cases}(a = \text{open}) : t-5 \\ (a = \text{close} : t+7)\end{cases}\right] \tag{9}$$

The observation gives a hint as to whether the temperature is low or high and its probabilities are given by

$$p(o = \text{high}|t', a = \text{open}) = \begin{cases}t \le 15 : 0.1 \\ t > 15 : 0.9\end{cases} \quad , p(o = \text{high}|t', a = \text{closed}) = \begin{cases}t \le 15 : 0.1 \\ t > 15 : 0.9\end{cases} \tag{10}$$

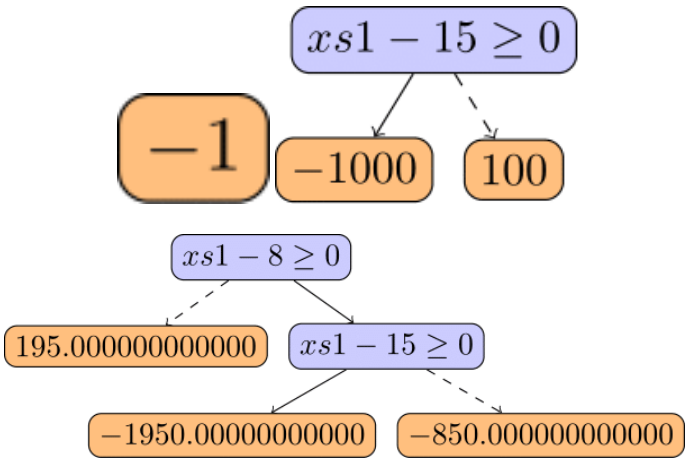

Figure 5: XADD representations of alpha functions computed at different points in the experiment (xs1 represents temperature).

The rewards are given by,

$$R(t,a) = \begin{cases} a = (\text{open}) & : -1 \\ a = (\text{closed}) \text{ and } t \geq 15 & : -1000 \\ a = (\text{closed}) \text{ and } t < 15 & : 100 \end{cases} \tag{11}$$

Three initial belief points were supplied to the algorithm: one where there is equal probability of the temperature being 5, 10, or 16, one with there being equal probability of the temperature being 5 or 10, and one where the temperature is known to be 5. The algorithm successfully computed backups and PBVI produced a quality solution to the problem. Figure 5 gives an example of what the different alpha functions start to look like at different time steps. The one negative result is that the computation time was very long even for such a simple problem suggesting that this does not scale well. Further experiments and theoretical analysis should try to evaluate the time-complexity of this algorithm precisely.

9 Conclusion

Ultimately, this algorithm proves to be useful for solving arbitrary H-POMDPs given initial belief points. It does however struggle to prove to be computationally efficient even on small H-POMDPs. It is difficult to see this algorithm scaling to work well on large optimization problems. Future work should focus on finding optimizations that can be applied to the algorithm to reduce its time-complexity. Also, the H-POMDP is a very special case of the broader problem of solving continuous POMDPs and it would be desirable for future work to extend this algorithm to have broader applications. First, extending this to deal with continuous action spaces as well as observation and state spaces would be a useful development. Also, developing an automated approach to approximating any continuous POMDP with an H-POMDP seems feasible and would greatly increase the applications of this algorithm. Perhaps, an approach could even be used to approximate the dynamics of an unknown model in the form of an H-POMDP which can then be solved with this algorithm. Also, future work should focus on analyzing the theoretical bounds for this algorithm and the computational complexity of solving arbitrarily complicated H-POMDPs.

References

[1] J. Pineau, G. J. Gordon, and S. Thrun. Anytime point-based approximations for large pomdps. *J. Artif. Intell. Res.*, 27:335–380, 2006.

[2] S. Sanner, K. V. Delgado, and L. N. de Barros. Symbolic dynamic programming for discrete and continuous state mdps. In *UAI*, 2011.

[3] Z. Zamani, S. Sanner, P. Poupart, and K. Kersting. Symbolic dynamic programming for continuous state and observation pomdps. In *NIPS*, 2012.