Using XADDs and Symbolic Dynamic Programming to solve POMDPs - Please do not peer review yet, almost done!

Justin Dieter
Email: jdieter@stanford.edu

***Abstract*—In many situations, it is desirable to produce an exact solution to a Partially Observable Markov Decision Process (POMDP) with continuous state and observation given initial belief points. Point Based Value Iteration (PBVI) has been extremely successful in producing approximate point based solutions. 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.**

I. Introduction

PBVI has proven to be successful as approximating point based backups for large scale POMDPs [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 [2]. An open source implementation of this algorithm in Julia is included alongside this paper and can be accessed at https://github.com/jdietz31/SDPPOMDP. Also, this paper gives a survey of the workings of the algorithm as well as an analysis of its performance on several sample POMDPs.

II. 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 [3]. 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 continuous function can theoretically be represented by an XADD,

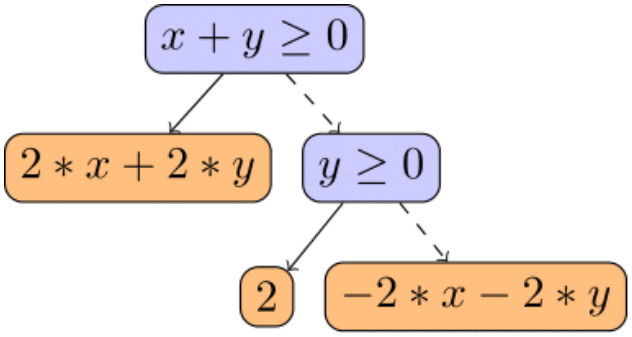

Fig. 1. An XADD version of the piecewise function in equation (1)

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

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

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

A. 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

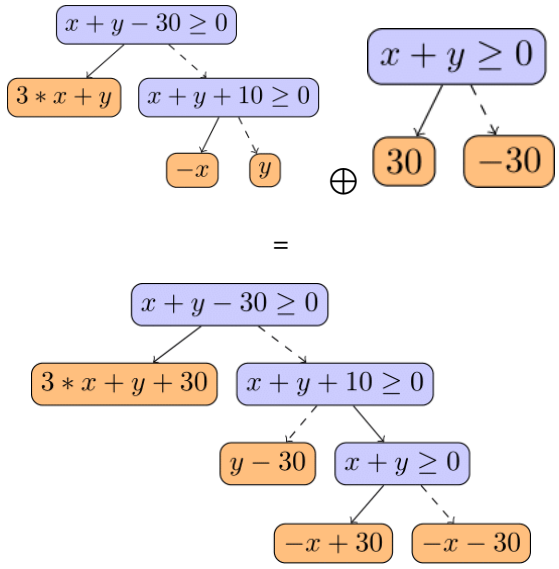

Fig. 2. An example of XADD addition (multiplication, max, and min work similarly)

is pruned. There are however, several optimizations in the algorithm to make it tractable. [3]

B. 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.

1) Binary Operations: Addition or multiplication ($\bigoplus$, $\bigotimes$) 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. [3]

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

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

Fig. 3. Outline of the standard PBVI algorithm

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:

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

III. 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.

IV. 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 $\alpha_j \in \Gamma^{h-1}$

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

$$\Gamma_a^h = R(s,a) + \gamma \boxplus_{o \in \mathcal{O}} [\{g_{a,o,j}^h(s)\}_j] \quad (4)$$

$$\Gamma^h = \bigcup_a \Gamma_a^h \quad (5)$$

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.

V. 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 $R : S \times A \rightarrow \mathbb{R}$ must be given by a piecewise linear function that can be represented by an XADD. Transition probabilities T and observation probabilities O are represented by a Bayesian network:

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

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

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.

VI. Conclusion

The conclusion goes here.

References

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

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

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