R12/Ishank Juneja/16D070012

March 11, 2020

Introduction

Under the classic Markov Decision Process (MDP) setup, we associate a tuple (S, A, T, R) with every MDP. In this notation, S is the set of states the agent can take, A is the set of possible actions for the agent, $T : S \times A \to \Pi(s)$ is the state transition function of the problem. T specifies the probability distribution over next states $s' \in S$ on taking an action $a \in A$ starting from state $s \in S$. R is the reward function which can be viewed as a feedback on agent performance from the environment. An assumption implicit to this model is the observability of state $s \in S$. Often times, it is not the case that an agent has access to its exact state. Instead, the agent might only have access to an observation $o \in \Omega$ which captures incomplete information about its underlying state. This framework satisfactorily models many practical situations. For instance, consider a robot that is attempting to localize its position (directly unobservable state s) in a multi-storied building by merely looking at a particular intersection in a corridor (observation o). Partially Observable MDPs (POMDPs) provide a richer mathematical framework which is able to describe many such situations.

The POMDP Framework

A POMDP is completely described by the tuple (S, A, T, R, Ω, O), where (S, A, T, R) is the underlying MDP, Ω is a finite set of observations the agent can experience and $O : S \times A \to \Pi(\Omega)$ is the observation function which gives, for every state and action, a probability distribution over possible observations. Under the POMDP framework an agent is unable to discern its state accurately. Instead of the true state, the agent makes "an observation" o based on the action a taken and the (still unobserved) resulting state s'.
An implication of this limitation is that an agent wishing to act optimally must use its memory of all previous actions and observations. To simplify the problem, the POMDP model introduces a *belief state* b. Belief states are probability distributions over the true underlying states of the agent. That is, for each state s, $b(s)$ gives the probability of the agent being in that state. By design, the agents belief state at any point of time is a sufficient statistic (SS) for all its previous experience. As a SS, the belief states capture all the past observations and the belief state that the agent started with. Sequential decision-making in

the POMDP becomes Markovian in belief state b due to the SS property.
To compute and update belief states as new experience is accumulated, the agent includes a block known as the state-estimator (SE). More precisely, given an observation o, the action taken a and the previous belief state b, the SE is the function $\text{SE}(b, a, o)$ which outputs the new belief state b'. Using the rules of conditional probability and the definitions of the POMDP parameters, the paper derives the relation

$$b'(s') \propto O(s', a, o) \sum_{s \in S} T(s, a, s') b(s),$$

where the probabilities in b' are computed using the constraint $\sum_{s \in S} b(s) = 1$.
The other necessary component to describe a POMDP agent is the policy being followed by it. In the POMDP framework, a policy maps a belief state vector b to an action a. Since the problem is Markovian in the belief state b, obtaining a policy through POMDP planning is equivalent to solving the planning problem for an associated **belief MDP**.
The parameters for the belief MDP are described below,

- $\mathcal{B}$ is the set of possible belief states and forms the state space
- The set of actions A remains the same as the POMDP
- $\tau(b, a, b')$ is the state-transition function which can be computed using the POMDP parameters
- $\rho(b, a)$ is the reward function and is given by an expectation over the reward function $R(s, a)$ under the distribution over states given by $b(s)$

Planning for POMDPs

It is often intractable to solve the planning problem for continuous state MDPs, however the special structure of the belief MDP lends it properties that give rise to efficient planning algorithms. In this paper the authors focus primarily on two planning approaches - **Exhaustive Enumeration and Witness Algorithm** the latter among which is a novel approach. Both methods provide a means to determine the optimal t - step value function V_t given a start belief state b. Here t - step refers to the situation where there are only t steps remaining in the agents lifetime. The paper considers the more general finite horizon problem since an infinite horizon discounted trajectory can always be approximated to arbitrary precision by a long enough finite trajectory. Lastly, once the optimal value function is computed using Value-Iteration, it is a straightforward task to determine the optimal policy to complete planning.
Next, the paper describes how a t-step policy can be represented using a *policy tree* - p of depth t. An expression for the expected return $V_p(s)$ associated with executing the policy p, starting from a state s, can be derived using the belief MDP parameters. However, the actual quantity of interest is the return for a given *belief state* b. Using the linearity of expectations we have, $V_p(b) =$

$\sum_{s \in S} b(s)V_p(s)$. Treating b and V_p as vectors of length $|S|$, V_p becomes a dot product.
The objective of planning through value-iteration is to obtain

$$V_t(b) = \max_{p \in \mathcal{P}} b \cdot V_p.$$

Pruning the Search space

Using geometrical arguments the authors were able to show that the optimal value function V_t is piecewise linear and convex. Such a V_t induces a partition on the space of possible belief states such that for every b there exists an optimal policy tree. While iterating over value functions, we would be much better of if our search space were as compact as possible. To do achieve this, we keep only *useful* policy trees and prune way the others. A policy tree p' is considered useful if its value function $V_{p'}$ is such that it cannot be dominated everywhere by the maximum of the value functions of any subset of $\mathcal{P}$. A possible way of ensuring that we have a minimal subset is to ensure that for every policy, there is at least one point in the state space where its own value function dominates. Once a minimal representation (set of policies) $\mathcal{V}_t$ for the optimal value function V_t is obtained, value iteration (VI) can be applied to the problem in a manner identical to that of VI for **discrete** MDPs.
The only problem left to be solved then is that of obtaining a minimal representation of the t step optimal value function given a minimal representation of the $t-1$ step value function V_{t-1}. The next two algorithms achieve this.

Exhaustive Enumeration

This algorithm works by constructing a superset $\mathcal{V}_t^+$ of the minimal set and then pruning it down to $\mathcal{V}_t$. The algorithm can be broken down into two steps - generation and pruning. To generate the superset $\mathcal{V}_t^+$, we only need to consider the policy trees contained in $\mathcal{V}_{t-1}$ since a policy with a non-useful subtree cannot itself be useful. This means that the superset will contain $|A||\mathcal{V}_{t-1}|^{\Omega}$ elements. To prune the superset, we can identify the optimal parent t length policy for each $t-1$ length sub-policy using linear programming. For this we would need to compute the value functions for each of the policy trees in $\mathcal{V}_t^+$. This can be done efficiently using the value functions of the sub-trees.
The main drawback of this approach is that we need to go through the superset $\mathcal{V}_t^+$ which has a size exponential in the number of possible observations $|\Omega|$.

Witness Algorithm - Authors' Novel Approach

Instead of computing $\mathcal{V}_t$ directly, the witness algorithm computes a set $\mathcal{Q}_t^a$ for every action a. $\mathcal{Q}_t^a$ represents the set of all t step policy trees with the action a at their root. The set $\mathcal{V}_t$ is then computed by pruning the union of such policies trees - $\cup_a \mathcal{Q}_t^a$. The job of the witness algorithm is to provide an *efficient* technique to compute $\mathcal{Q}_t^a$. In particular the authors show that their algorithm

has complexity polynomial in $|S|, |A|, |\Omega|, |\mathcal{V}_{t-1}|$ and $|\mathcal{Q}_t^a|$. A caveat here is that $|\mathcal{Q}_t^a|$ can be exponentially larger than the size of $\mathcal{V}_t$.
The core of the witness algorithm lies in its pruning technique. The algorithm starts of with a single policy tree optimal for some arbitrary belief state b. Then we check if there exists a belief state b for which the Q value estimated using our current set of trees - $\hat{\mathcal{Q}}_t^a(b)$ is less than the true Q value given by $\mathcal{Q}_t^a(b)$. Such a b is said to serve as a *witness* to the fact that our current collection of policy trees is incomplete. Once such a witness is identified, the algorithm appends the policy, with action a at the root, that yields the best value for the earlier chosen belief state b. This process continues until no more witness points exist and we conclude that our representation of Q is perfect. Since the belief space is continuous, the search for witnesses b is achieved using linear programming.
The witness algorithm fails to perform well on problems where $|\mathcal{Q}_t^a|$ is "not polynomial". In such situations the authors point to other algorithms such as the briefly described *linear support algorithm.*

Obtaining a Minimal Finite State Controller

For some special POMDP problems, the optimal policy trees have the property that the observation-action mapping at every level remains the same. In such situations the policy becomes stationary and can be represented as a more compact plan-graph. The way a plan graph is created, on every step an agent could start of in the node optimal for its initial belief state and thereafter simply make observations and follow the arc associated with the observation to the next node. This picture is essentially that of a finite state controller which uses the minimal possible amount of memory to act optimally in a POMDP environment. As the paper points out, it is fascinating that the POMDP problem started off as a discrete problem, was converted to a continuous belief space and at the end the continuous solution could again be mapped back to a discrete controller.

Some questions I have are -

- What exactly is the efficient pruning strategy for obtaining the minimal useful set $\mathcal{V}$ of policy trees? It seemed to me that we can't do without ensuring that every policy we throw out does not dominate all other policies for some $b \in \mathcal{B}$.
- In the description of the witness algorithm (WA), the authors claim that the running time of WA is polynomial if $|\mathcal{Q}_t^a|$ is polynomial. What is meant by the latter being polynomial? What problem parameters are being refereed to?