Tutorial on Markov Decision Processes

Fernando S. Barbosa (fdsb@kth.se)

1. Introduction

This tutorial is part of the Probabilistic Graphical Models (PGM) PhD course and it is designed to cover theory and application of a specific type of Bayes Network used for decision making: the Markov Decision Processes (MDP). The main goal of this tutorial is to guide the reader through the theory behind MDPs and Partially Observable Markov Decision Processes (POMDPs) and implement both on a Grid World problem. Online POMDP solving is left as optional for interested students.

The main references used on this tutorial are:

- Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- Kochenderfer, Mykel J. *Decision making under uncertainty: theory and application*. MIT press, 2015.
- Russell, Stuart J., and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

2. Markov Decision Processes (MDPs)

2.1 Background

Markov Decision Processes rest on two main assumptions, (i) the system state is fully observable, i.e. it has access to all information during all the time without any noise, and (ii) the transition model is Markovian, i.e. the probability of reaching state *s'* from *s* depends only on *s* and not on the history of previous states. MDPs treat only one type of uncertainty: the one related to the robot's motion. Formally, an MDP is defined as a tuple (S, A, T, R), where S is a set of states, A the set of actions, T the transition model (e.g. $P(s' \mid s, a)$) and R the reward function. It can be represented as a decision network as in Figure 1.

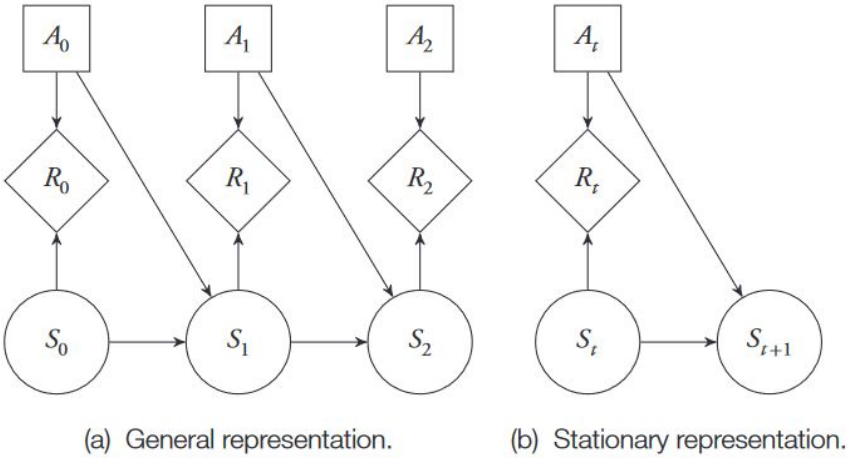

Figure 1: MDP decision diagram

Suppose an environment such as the one presented in Figure 2(a), in which the robot is deployed from cell (1,1) and the available actions are *Up*, *Left*, *Right*, *Down*. Its interaction ends as soon as it reaches any of the goal cells, marked with reward +1 or penalty -1. In the case of deterministic robot action, the solution is straightforward: {*Up*, *Up*, *Right*, *Right*, *Right*}. However, this might not solve the problem if the transition model is stochastic, as in Figure 2(b). In this case, the robot will follow the given command with probability 0.8, but it might take left or right with probability 0.1 each.

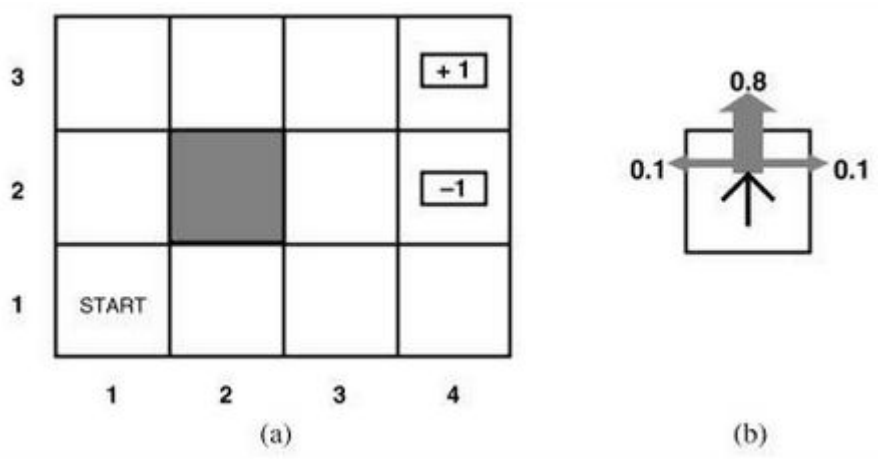

Figure 2: A simple 4x3 environment in (a) and its transition model in (b).

Exercise 1: *Given Figure 2 and the straightforward solution {Up, Up, Right, Right, Right}, calculate which cells can be reached and with what probabilities.*

The solution to an MDP problem is a mapping from states to actions, usually called (control) policy and denoted by □. An optimal policy π^* is such that it yields the highest expected utility of the possible histories generated by that policy. Figure 3 presents an optimal policy for the previous example taking into account $R(s) = -0.04$ for nonterminal states.

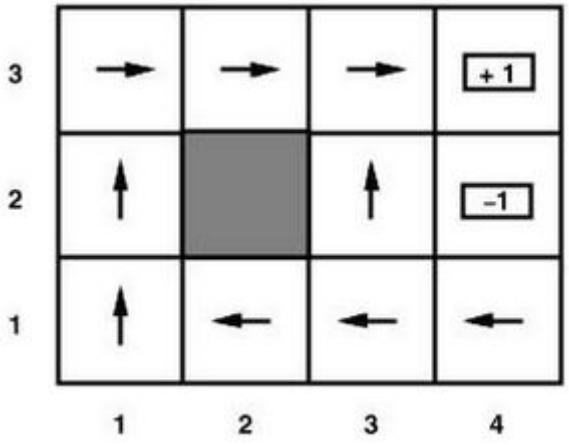

Figure 3: An optimal policy for the Figure 2 MDP.

2.2 Value Iteration

Value Iteration is the most common algorithm for finding optimal policies. It recursively calculates the utility of each action relative to a reward function. The first step when deciding on the utility function is to determine its *planning horizon*. There are basically three cases:

1. Greedy case: the robot only seeks to minimize the immediate next penalty.
2. Finite-horizon case: there is a fixed time N after which nothing matters. The optimal policy for such cases are *nonstationary*, i.e. they vary according to the given planning horizon.
3. Infinite-horizon case: there is no time limit, so there is no reason to behave differently in the same state at different times. The solution is called *stationary*.

Assuming that the robot is in some initial state s and defining S_t, a random variable, as the state it reaches at time t when executing π, the expected utility obtained by executing π starting in s is given by

$$U^{\pi}(s) = E\left[\sum_{t=0}^{N} \gamma^t R(S_t)\right], \tag{1}$$

a expectation with respect to the probability distribution over state sequences. γ is known as the *discount factor*, a problem-specific parameter constrained to the interval $[0;1]$. This discounted reward formulation allows the infinite-horizon case to have a finite utility. In fact, if $\gamma < 1$ and $\left|R(S_t)\right| \leq R_{max}$,

$$\sum_{t=0}^{\infty} \gamma^t R(S_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma). \tag{2}$$

The optimal policy is the one that, out of all policies the robot could choose to execute, will have higher expected utilities than others. Formally,

$$\pi_s^* = \arg\max_{\pi} U^{\pi}(s). \tag{3}$$

The utility function $U(s)$ allows the robot to select the action that maximizes the expected utility of the subsequent state:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s'). \tag{4}$$

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action, as described by the *Bellman equation*:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s'). \tag{5}$$

The *Value Iteration* algorithm approximates the Bellman equation with an iterative approach. The sketch of the algorithm is presented in Figure 4.

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, the maximum change in the utility of any state in an iteration

repeat
$U \leftarrow U'$; $\delta \leftarrow 0$
for each state s **in** S **do**
$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

Figure 4: The value iteration algorithm.

Exercise 2: *How would you write the Bellman equation if the MDP were formulated with a reward function* $R(s,a)$ *that depends on the action taken? And if it also depends on the outcome state* $R(s,a,s')$ *?*

2.3 Implementation

The file *PGM_MDP.m* contains a basic implementation of the Grid World problem. Given the map and all the required parameters, it outputs the optimal policy for both deterministic and stochastic robot action cases.

There is one line of code missing, specifically line 159. It corresponds to part of the Bellman equation. Based on the previous section and on line 142 (the deterministic implementation), complete this line of code and run some simulations. Assume that the robot has probability *probStraight* of going straight but it might end up in either to the left or to the right of the desired tile.

Explore the code, make sure to understand how it works and is processed, try to relate each part to the algorithm described here. Run simulations with different penalties and action probability and observe how the result changes. You need familiarity with the code to implement the POMDP in the following section.

3. Partially Observable Markov Decision Processes (POMDPs)

3.1 Background

Due to sensor limitations or noise, the robot's state might not be perfectly observable, giving rise to Partially Observable MDPs, a sequential decision problem that takes into account state uncertainty. Partial observability implies that the robot has to estimate a posterior distribution over possible world states. Furthermore, the utility of a state and its

optimal action depend not just on the state itself, but also on the belief of being there. Therefore, the optimal policy is a mapping from belief states to action.

A POMDP has the same structure as an MDP, with the addition of the Observation model, usually written as the probability of observing o given state s, i.e. $O(o \mid s)$. Figure 5 shows the POMDP problem as a dynamic decision network in which the robot does not have direct access to the current state. Figure 6 outlines a basic POMDP policy execution.

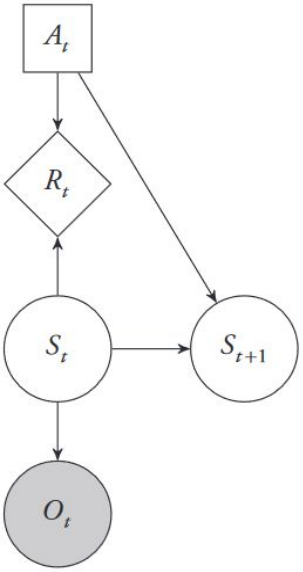

Figure 5: POMDP problem structured as a Dynamic Decision Network.

Algorithm 6.1 POMDP policy execution

```
1: function POMDPPolicyExecution(π)
2:     b ← initial belief state
3:     loop
4:         Execute action a = π(b)
5:         Observe o and reward r
6:         b ← UpdateBelief(b, a, o)
```

Figure 6: Outline of a POMDP policy execution.

Let us calculate the probability that a robot in belief state b reaches b' after executing action a:

$$P(b' \mid b, a) = \sum_o P(b' \mid b, a, o)\, P(o \mid b, a) \tag{6}$$

$$= \sum_o P(b' \mid b, a, o) \sum_{s'} P(o \mid b, a, s')\, P(s' \mid b, a) \tag{7}$$

$$= \sum_o P(b' \mid b, a, o) \sum_{s'} O(o \mid s') \sum_s P(s' \mid s, a)\, b(s)\,. \tag{8}$$

The immediate reward is given by

$$R(b,a) = \sum_s R(s,a)b(s)\,. \tag{9}$$

In problems with a discrete state space, recursive Bayesian estimation can be applied. Suppose that the initial belief state is b, from which o is observed after executing action a. The new belief state s' can be written as follows

$$b'(s') = P(s' \mid b,a,o) \tag{10}$$

$$\propto P(o \mid s',a,b)\,P(s' \mid b,a) \tag{11}$$

$$\propto O(o \mid s',a)\,P(s' \mid a,b) \tag{12}$$

$$\propto O(o \mid s',a)\,\sum_s P(s' \mid s,a)b(s)\,. \tag{13}$$

In (8), the probability of being in b' given b,a,o is 1 if it is equal to the result from (13), and 0 otherwise.

Exercise 3: *This exercise aims to clarify the belief state update method described in (13). Suppose the following crying baby problem. You are assigned to feed a baby. The observed variable is the cry of the baby, since it is not possible to know for sure if it is actually hungry or not. There is a 10% chance the baby cries when not hungry, and 80% it cries when hungry. The dynamics are as follows. If you feed the baby, then it is not hungry anymore at the nest time step. If the baby is not hungry and we do not feed it, then there is a 10% probability it may become hungry at the next time step. Once hungry, the baby keeps hungry until fed. Based on these facts, answer the following:*

a) *Specify the variables of the model and write the conditional probabilities of the observational and the transition models.*
b) *Assume the initial belief state is* $b(h^0, h^1) = (0.4,\ 0.6)$*, i.e. there is some prior knowledge that indicates a slightly higher chance of the baby being hungry. What would be the new belief state if you decide not to feed the baby and it ends up crying?*
c) *Next, since the baby is crying, you decide to feed it. What is the new belief state if the baby stops crying?*

3.2 Offline Method

Offline POMDP solution methods involve performing most of the computation prior to execution. The most used and straightforward is QMDP, a hybrid between MDPs and POMDPs. It generalizes the MDP-optimal value function defined over states into a POMDP-style value function over beliefs by assuming that all state uncertainty disappears at the next time step.

The QMDP algorithm is as follows. Prior to the execution, one executes the MDP value iteration algorithm as in the previous section. Based on the outcome, i.e. on the optimal control policy assuming full observability, the robot applies the algorithm outlined in Figure 6 with line 4 given by:

for all control actions u do:

$$Q(x_i, u) = r(x_i, u) + \sum_{j=1}^{N} \hat{V}(x_j) p(x_j | u, x_i)$$

endfor

return $\arg\max_u \sum_{i=1}^{N} b(x_i) Q(x_i, u)$

3.3 Implementation

Implement the previous offline POMDP algorithm on the same environment from the last implementation exercise. For the observation model, assume the robot has probability *probObserv* of observing the correct location and *(1-probObserv)* distributed around it.

Hint: use the result from the MDP implementation to calculate $Q(x_i, u)$.

4. Online POMDP Solving [Optional]

Solving POMDPs optimally is computationally intractable, and depending on the size of the problem, even approximating the optimal solution is difficult. The two main challenges are the curse of dimensionality and the curse of history, resulting in exponential growth of computational complexity. Online methods determine the optimal policy by planning from the current belief space, usually interleaving planning and execution. The belief states reachable from the current state are typically small compared with the full belief space.

Many online methods use a depth-first tree-based search, the belief-tree. The current belief is set as the root of the tree, and the agent performs lookahead search on the tree for a policy that maximizes the expected total discounted reward. Each node represents a belief and branches into $|A|$ action edges, which further branches into $|Z|$ observation edges.

The approximately optimal policy is obtained by truncating the tree at a predefined depth. At each leaf node a default policy is simulated to obtain a lower-bound estimate on its optimal value.

The current state-of-the-art online solver is DESPOT[1], *Determinized Sparse Partially Observable Tree*. It performs online planning on randomly sampled scenarios and returns a near-optimal policy. Figure 7 briefly compares a DESPOT search tree with a standard lookahead tree.

In case you enjoyed implementing MDPs and QMDPs and feel like trying out some online planning, check out DESPOT's open-source project in GitHub.

[1] Ye N, Somani A, Hsu D, Lee WS. *DESPOT: Online POMDP planning with regularization*. Journal of Artificial Intelligence Research. 2017 Jan 26;58:231-66.

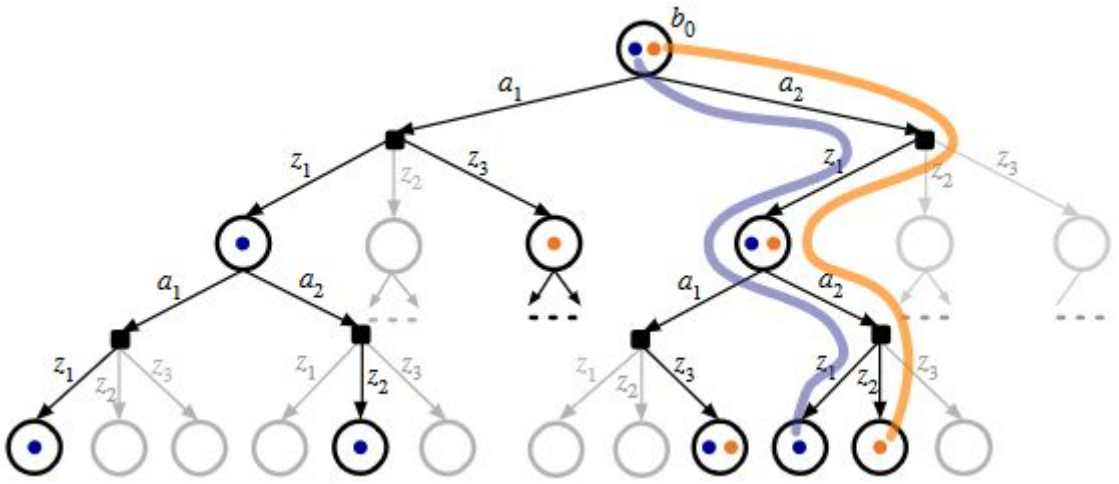

Figure 7: In black, a DESPOT tree. In light gray a traditional lookahead search tree.

5. Conclusion

This tutorial guided the reader through Markov Decision Processes without going too deep into theory. Based on the Value Iteration algorithm, the text presents the main equations and pseudo-algorithms, besides a few conceptual exercises and simulation implementation.