POMDPs: Myths, Legends, and Reality

John Biechele-Speziale, Kushagra Kapoor, Jae Heo
May 24, 2023

PURDUE UNIVERSITY

Outline

What are POMDPs?

POMDPs are MDPs with sensors instead of direct observation of the current state.

- The sensors are some probability distribution of some observation given a certain state.

POMDPs are MDPs with sensors instead of direct observation of the current state.

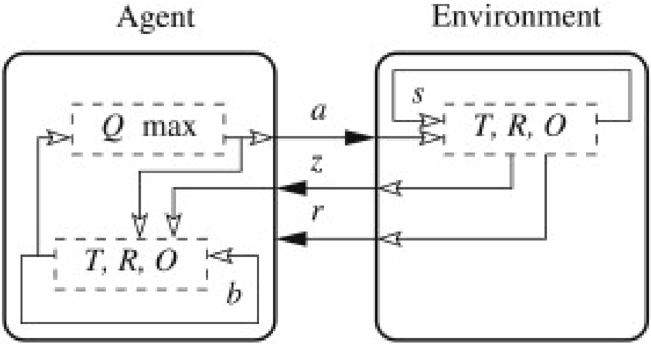
- The sensors are some probability distribution of some observation given a certain state.
- Unlike traditional MDPs that map from states to actions, these map from belief states(observations) to actions.

POMDPs are MDPs with sensors instead of direct observation of the current state.

- The sensors are some probability distribution of some observation given a certain state.
- Unlike traditional MDPs that map from states to actions, these map from belief states(observations) to actions.
- Exact optimal solutions yield the optimal action for each possible belief state that minimizes our cost.

POMDPs are MDPs with sensors instead of direct observation of the current state.

- The sensors are some probability distribution of some observation given a certain state.
- Unlike traditional MDPs that map from states to actions, these map from belief states(observations) to actions.
- Exact optimal solutions yield the optimal action for each possible belief state that minimizes our cost.



POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions
- T is the state transition probability $P(s'|s, a)$

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions
- T is the state transition probability $P(s'|s, a)$
- R is the reward function $R(s, a)$

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions
- T is the state transition probability $P(s'|s, a)$
- R is the reward function $R(s, a)$
- Ω is the set of observations

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions
- T is the state transition probability $P(s'|s, a)$
- R is the reward function $R(s, a)$
- Ω is the set of observations
- O is the set of conditional observation probabilities $P(o|s') \vee P(o|s', a)$

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions
- T is the state transition probability $P(s'|s, a)$
- R is the reward function $R(s, a)$
- Ω is the set of observations
- O is the set of conditional observation probabilities $P(o|s') \vee P(o|s', a)$
- γ is the discount factor.

POMDPs are defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$

- S is the set of states
- A is the set of actions
- T is the state transition probability $P(s'|s, a)$
- R is the reward function $R(s, a)$
- Ω is the set of observations
- O is the set of conditional observation probabilities $P(o|s') \vee P(o|s', a)$
- γ is the discount factor.
- As usual, we want an optimal policy (π) that maximizes expected future reward.

POMDPs rely on a unique belief update step as a part of the algorithm.

- a

Notes on theory, undecidability, and intractability

-

POMDP Variants and Applications

Various methods of solving/approximating POMDPs are currently available/used.

- On Policy Methods
  -
- Off Policy Methods
  -

Implementation Recommendations

My favorite package: POMDPs.jl is my favorite because it's easy to use: Installation

```
using Pkg; Pkg.add("POMDPs"); Pkg.add("QMDP");
using POMDPs, QuickPOMDPs, POMDPTools, QMDP
```

My favorite package: POMDPs.jl is my favorite because it's easy to use: Definitions

```
m = QuickPOMDP(
    states = ["left", "right"],
    actions = ["left", "right", "listen"],
    observations = ["left", "right"],
    initialstate = Uniform(["left", "right"]),
    discount = 0.95,
    transition = function (s, a)
        if a == "listen"
            return Deterministic(s) # tiger stays behind the sam
        else # a door is opened
            return Uniform(["left", "right"]) # reset
        end
    end,
    observation = function (s, a, sp)
        if a == "listen"
            if sp == "left"
                return SparseCat(["left", "right"], [0.85, 0.15]
            else
```

Alternative Packages

- Finite-state Controllers using Branch and Bound
- pomdp
- pyPOMDP
- zmdp

Temp 1

Temp 2

Temp 3

Temp 4

Temp 5

References I